Amazing Computing™

Vol. 1/ Number 7
U.S.A. $3.50
Canada $4.50

Commodore Amiga™ Information and Programs

TWO Complete 3D Graphics Programs

Amazing Dealers

The following are **Amazing Dealers**, dedicated to supporting as well as selling the **Commodore-Amiga™**. They carry **Amazing Computing™**, your resource for information on the Amiga™.

If you are not an **Amazing Dealer**, but would like to become one, contact:

PiM Publications, Inc.
P.O.Box 869
Fall River, MA. 02722
1-617-679-3109

Alabama
Abax Data Systems — Huntsville
Command Computers — Birmingham
Madison Books & Computers — Madison
Mel's Photo & Computer Shop — Montgomery
Universal Computer System — Mobile

Arizona
Computer West — Phoenix
Copperstate Business Systems — Phoenix
North American Digital — Tucson
Tronixtools — Tucson

Arkansas
SIS Inc. — Mt. Home
The Micro Shop — Little Rock

California
Brown Knows Computing — Redlands
C.F.T. — Claremont
Century Computer Systems — La Habra
Chambers Computer Supply — Studio City
Computer Attic — Palo Alto
Computer Junction — Santa Ana
Computer Literacy Bookshop — Sunnyvale
Computer Nook, Inc. — San Bernadino
ComputerLand Freemont — Freemont
ComputerLand of Stockton — Stockton
Computer Technology — Pomona
Home Computing Center — San Bruno
K J Computers — Granada Hills
Levity — Hollywood
Op Amp Tech Books — Los Angeles
Ridgecrest Computer Ctr. Inc. — Ridgecrest
S.O.S. Computers — Los Angeles
Software First — Santa Rosa
Software Plus — Citrus Heights
The Computer Room — Scotts Valley
The Floppy Disk Inc. — Downey
Winner's Circle — Berkley
HT Electronics — Sunnyvale

Colorado
Citadel Computers — Colorado Springs
Computer Discount — Denver
Computer Room — Aurora
Ideal Computer Systems — Evergreen
Micro World — Lakewood
Sun Country Computers — Colorado Springs
Whole Life Distributors — Englewood

Conneticut
Connecting Point — Georgetown
Mnemonics Inc. — Norwalk
Personal Computer Center — Norwich
Software Kingdom — East Windsor
Spectrum Computers — New Haven

Delaware
Castle Video — Newark

Florida
A A Computers — Jacksonville
Book Mania — Orlando
Commodore Computer Shop — Ocala
Computer Bar — Pennsacola
Computer Image — Miami
Computer Terminal Inc. — Sebring
Computer Ware Inc. — Tamarc
Computers 4 Rent — W. Palm Beach
Computers Plus — South Daytona
Cotronics Inc. — Stewart
Education Computers Etc. — Tallahasse
Family Computers — Sarasota
Florida Book Store — Gainesville
Gulf Coast Computer Elect — Panama City
Megaport Computer Center — Venice
Micro's Etc. — Altamonte Springs
MSI Business System — Brooksville
New Age Electronics — St. Petersburg
Rainbow Computer Center — Orlando
Random Access Computers Inc. — Ft. Walton Beach
Softwalls Computer Center Inc. — Holly Hil
The Open Door — Cocoa
The PC Collection — Coral Gables

Georgia
Future System — Augusta
Micheal Donnellan — Decatur
Software South Inc. — Savannah
The 64 Store — Atlanta

Hiwii
Computer House Inc. — Honolulu

Iowa
Century Systems — Des Moines
Micro Computer Applications LTD — Marshalltown

Idaho
ABI Computer & Video — Nampa

Illinois
BA Business Computers — Batavia
Computer Resolution — Champaign
Computerland O'Hare — Park Ridge
ComputerLand of Niles — Niles
Illini Microcomputers Inc. — Naperville
The Memory Expansion — Normal
Unique Computor — Sterling

Indiana
Burkat Computer Ctr. — South Bend
Bytrex Computer — Ft. Wayne
Computer Corner — Fort Wayne
Computer Corner — Marion
Computer People Inc. — Michigan City
Greg Chaney — Greenfield
Micro Computers Inc. — Indianapolis
Vons Computers — West Lafayette

Kansas
Data Bank Corp. — Wichita
J & L Electronics — Liberal
MidKansas Computers — Newton
Midwest Computers — Manhattan
Thorobred Computers — Topeka

Kentucky
MicroAge Computer Store — Lexington
Reality World Computers — Lexington

Louisiana
Modern Business Machines — Metairie
Software Center International — Metairie
Software Mart — Metairie
The Computer Clinic — Lafayette

Massachusetts
Club Computer — Cambridge
Computerland — Leominster
E.U. Wurlitzer Co. — Boston
General Computer Store — Framingham
HCS Computer Center — Marshfield
LCA Video & Computer Ctr. — Norwood
Memory Location — Wellsley
OmniTek Computers — Tewksbury
Pioneer Valley Data Equipment — Amherst
Tech Computer Store — Cambridge
The Bit Bucket — West Newton
Tycom Inc. — Pittsfield

Maryland
Computer Crafters — Wheaton
ComputerWorld — Ocean City
Compuvision Vision Ctr. — Columbia
Micro Computer Center — Baltimore
Software Advantage — Rockville
The Logical Choice Inc. — Baltimore
Waldorf Computer — Waldorf

Maine
Computer Barn — Damariscotta
Valley Computers — Auburn

Michigan
Basic Computer Center — Grand Rapids
Bits-Bytes-Nibbles — Petowskey
Chelsea Computer — Ann Arbor
Computer Supply Co. — Traverse City
Compusoft — Kalamazoo
Direct Access — Lansing
Edmunton Computer Center — Pontiac
Galaxy Computers — Allen Park
Global Computer Center — Dearborn
Michigan Software — Novi
Midland Computer Shop — Midland
Pro-Video — Okemos
Professional Computer Systems — St. Joseph
Programs Unlimited — Grand Rapids
Roseville Computer Store — Warren
Slip Disk — Madison Heights
Software Plus — Grand Rapids
Specialists Computer — Livonia
The Software House — Kalamazoo
Ye Olde Computer Shoppe — Ypsilanpi
Retail Computer Center Inc. — Farmington Hills

Minnesota
Computer Outfitters — Roseville
Computer Place — Minneapolis
Computer Specialties Plus — Monticello
"Computers, etc." — Eagan
JMH Software — Maple Grove
MCD of Hibbing Inc. — Hibbing
OnLine Computers — Minnetonka
Specialists IM — Hopkins

Missouri
Associated Computer Services — Springfield
Brands Mart Computers — Kansas City
Computer Concepts — Columbia
Instant Replay Ltd. — Chesterfield
Systems Plus — Independence

Mississippi
Enterprises Unlimited — Jackson

Montana
Appelgren Computer Systems — Great Falls

North Carolina
Digitz — Raleigh
TDS Computers — Carrboro
Triad Computers — Greensboro

North Dakota
Computer Associates — Fargo
The Computer Store — Minot

Nebraska
BULLS-EYE — Lincoln
Double E Electronics — Omaha
My Com Inc. — Plattsmouth

New Hampshire
Computer Mart of New Hampshire — Nashua
Diversified Computers — Keene

New Jersey
Family Computer Centres — Chester
Family Computer Centres — Ridgewood
Family Computer Centres — South Orange
Family Computing Centres — Ridgewood
Morris County Stationers — Chester

New Mexico
Academy Computers — Albuquerque
New Horizon Computer System — Alamogordo
Page One Newstand — Albuquerque
Technical Concepts — Las Crues

Nevada
Century 23 Inc. — Las Vegas
Computer World — Las Vegas
ComputerLand of Carson — Carson City

New York
"Amuse" — New York
Byte Shop — Merrick
Castle Computer — Latham
CIA Software Center — Flushing
Compurite Center — White Plains
Computer Outlet — Jamestown
Leigh's Computers — New York
Ray Supply Inc. — Glen Falls
Software & Such — Scotia
Software City — Forest Hills
Software Supermarket — Kenmore
Star Tech Systems — Massena
Tedrow Business Products — Rochester
The Working Computer — Stoney Brook
Video Computer Center — Rome
World Computors Inc. — Hicksville
Computers Etc. — Syracuse

Ohio
Computer Network — Newark
Computers Plus of Ohio — Upper Sandusky
Earthrise Micro Systems — Columbus
Earthrise Micro Systems — Delaware
Lakes Consumer Electronics — Akron
Micro Center — Columbus
Microwave Magic — Fairfield
North Coast Programming — Willoughby
Quality Computer Applications — Toledo
Quality Computer Applications — Toledo
Saxtan Computer Center — Sandusky
Software Centre International — North Olmsted
Software and More — Cincinnati
Software Connection — Mayfield Hts.

Oklahoma
Colonial Video & Computer — Bartlesville
P C Tech — Tulsa
Second Hand Software — Oklahoma City
Video-Comp Inc. — Lawton

Oregon
Clackamas Computers — Beaverton
Clackamas Computers — Clackamas
Computron Business Systems — Portland
Computron Business Systems — Tigard
IB Computers — Portland
Software Center — Beaverton
Software Express — Eugene
University of Oregon Bookstore — Eugene

Pennsylvania
Alpha Omega EBE — Warren
Basic Computer Systems — Hermitage
Data Sofique Computerware — Pittsburg
East Coast Software — Hershey
Pittsburgh Computer Store — Pittsburgh
The Electronic Boutique — Norristown
Triangle Computers — Indiana

Rhode Island
International Computer Services — Johnston
Micro Limits — Smithfield
Micro Limits — Warwick

South Carolina
Avcom International Inc. — West Columbia
Horizon — Greenville

South Dakota
Computer Business Center — Rapid City

Tennessee
ComputerLab of Memphis — Memphis
Software First — Nashville

Texas
Byte by Byte — Austin
C.E.M. Corporation — Brazoria
Colonial Video — Houston
Computer Age — Houston
Computer Age — Dallas
Computer Magic — Austin
Computer Revelations — Houston
Computer Time Inc. — Marshall
"Lee Kaplan, Metropolitan Computer" — Richardson
Micro Search — Houston
Regency Educational Systems — Dallas
Softex — Watauga
Software Centre — Dallas
Software Library — Witchita Falls
Software Terminal — Fort Worth
The Computer Experience — San Antonio
The Computer Stop — Abilene

Utah
Computers Plus — Murray

Virginia
Colony Ltd. — Hampton
Diskovery — Falls Church
Family Computer Center — Fairfax
Virginia Micro Systems — Woodbridge

Washington
A.P.P.L.E. CO-OP — Renton
Com-Sott Computers — Everett
Computers + Inc. — Tacoma
Micro Age Computer Store — Bellview
Programs Plus — Seattle

Wisconsin
Colortron Computers — Racine
Computer Software Center — Milwaukee
Mom's Computer — Stevensport
Software Shoppe — Fond Du Lac
Sound Creations — Madison
TMW Software Inc. — Wausau

Canada
Compute or Play — Alberta
Computerworks — Alberta
Computer Shop — Alberta
Data Pro — Alberta
Computer Shop of Calgary — Alberta
The Software House — Alberta
Hindsin Computing — Alberta
Red Deer Micro Systems — Alberta
Memory Land — Alberta
Computer King — British Columbia
Cont — British Columbia
Nutec — British Columbia
Sprite — British Columbia
Strider — British Columbia
Super Software — British Columbia
Canadian Computer Era — Manitoba
Micromart — Manitoba
ComputerWorld — Maritmes
J.W. Allen — Maritmes
Kobetek — Maritmes
Miniomp — Maritmes
Arkon — Ontario
Chatam Computer — Ontario
Computerland — Ontario
Comspec — Ontario
Electronics 2001 — Ontario
Hastings Data — Ontario
The Computer &You — Ontario
Thornhill — Ontario
Software Co — Quebec
Deltronic Solare — Quebec
Merodata — Quebec
Info Mereier — Quebec
Laval Micro — Quebec
Vision Tronique — Quebec
Distribution Visipro — Quebec

Distributed in Canada by
Avita Software Distributors — Concord, Ontario
Compulit Distributors — British Columbia
Phase 4 Dist. Inc. — Southeast Calgary
1-800-661-8358 — Alberta,

SoundScape...Power Play for the AMIGA.

Pro MIDI Studio

The most powerful performance and recording software on any computer. The recording studio-like environment provides complete facilities for routing, recording, editing, transposition and playback of any musical performance. As new modules are introduced, you can "install" them at any time. Music can be performed by the internal sampled sound synthesizer, or with any external MIDI equipment. Record from the QWERTY keyboard or any external MIDI source, including keyboards, guitar and pitch followers. Synchronize with, or provide MIDI clock information, including MIDI Song Pointers. The complete flexibility of the system makes your imagination the only limit to its power.

- Number of notes and tracks determined by available memory
- MIDI patch panel links program modules
- Install new modules at any time
- Up to 16 internal instruments at one time
- Complete sample system with editing, looping, ADSR envelopes, velocity sensitivity, and pitchbend.

- Up to 160 sampled sounds at one time
- Save and load IFF note and sample files
- Quantize to any multiple of MIDI clock beats
- "Match" mode eases learning of a song
- Complete MIDI sequence and song editing
- Route, merge, split, or bounce any track to any other.

MIDI Interface

Necessary for any program which supports MIDI to communicate with MIDI equipment.

- Completely compatible with the standard Amiga MIDI interface
- MIDI In, Out, and Thru connectors
- Plugs into the serial port

Sound Digitizer

With the SoundScape Sound Digitizer, any sound may be sampled and modified by the Amiga, including voice. IFF File compatibility enables these samples to be used as musical instruments, sound effects, or speech with any IFF compatible music or animation system.

- High quality
- Highest possible fidelity from the Amiga
- Stereo or mono
- Variable sample rates
- Mike and line inputs
- Digitally controlled volume on each channel
- IFF Sample File compatible
- Software included for sampling, editing, and MIDI performance functions

Available From Your AMIGA Dealer.

SoundScape Pro MIDI Studio	$149.00
AMIGA MIDI Interface	$ 49.00
SoundScape Audio Digitizer	$ 99.00

mimetics™

corporation ...the professional software source!!

P.O. Box 60238 Sta. A, Palo Alto, CA 94306 (408) 741-0117

Amiga is a trade mark of Commodore Business Machines

Prices and availability subject to change without notice

PiM PUBLICATIONS, Inc.

Amazing Computing™

Publisher: Joyce Hicks
Circulation Manager: Doris Gamble
Assistant to the Publisher: Robert James Hicks
Corporate Advisor: Robert Gamble

Managing Editor: Don Hicks
Hardware Editor: Ernest P. Viveiros Sr.
Amicus & Technical Editor: John Foust
Music Editor: Richard Rae
Assistant Editor: Ernest P. Viveiros Jr.

Assistant Advertising Manager: John David Fastino

Amazing Authors:
Ervin Bobo
Bryan Catley
John Foust
Don Hicks
Kelly Kauffman
Perry Kivolowitz
George Musser Jr.
Steven Pietrowicz
Rick Wirch
&
The Amigo

Special Thanks to:
Robert H. Bergwall
RESCO, Inc.
E.P.V. Consulting
New England Technical Services
Interactive Tutorials Inc.

Advertising Sales
&
Editorial

1-617-678-4200

Amazing Computing™ (ISSN 0886-9480) is published by PiM Publications, Inc., P.O. Box 869, Fall River, Ma. 02722. Subscriptions: in the U.S. 12 Issues for $24.00; Canada and Mexico, $30.00; Overseas, $35.00. Printed in the U.S.A.

First Class or Air Mail rates available upon request.

PiM Publications maintains the right to refuse any advertising.

Subscription Problems?
Moving?

Please let us know

send information to:
PiM Publications, Inc.
P.O. Box 869
Fall River, MA. 02722

If we don't know where you are, we can not send you your magazine!

Amazing Computing

Table of Contents

Volume 1, Number 6 1986

The AMICUS Network

Departments:

From The Editor:

Birth of a Cover:

One of the more difficult tasks at the magazine is the design of our cover. Oh, I know. I can hear you muttering that our simple covers could not possibly take a great deal of effort. They are so simple, sometimes misspelled, and for the first few issues, the same color.

Well, although simple, our covers are a result of hours of attempts and failures at catching just the right concept. We push and learn what our printer can do and what we can accomplish.

Last issue we began publishing AC with a four color process cover (this is the same as some of the "bigger" magazines). We were responding to the cry from advertisers for color space in the magazine. We were certain that the layout would be simpler, now we could just take a picture and let the printer do the separations. Yeah!

It didn't work that way. It seems the more you can do, the more you want to do.

We now had the tools, but we were not content. We felt we should push a little harder. Each member of the staff had their own input as to what we should do for the cover. No one was extremely satisfied with any of the others' ideas.

Therefore, in the middle of the night, I took their suggestions put them on the cover and, in the morning, quietly gave them to the printer. Everyone's reactions? "Well, it was OK."

So I vowed this cover would be better, stronger, and with a lot less dessention. However, I made one mistake. This issue was intended as the graphics issue and the cover should utilize some of the tools and equipment available to the Amiga artist. Great.

The Amiga is not quite a year old, but do you know how many tools their are for graphics? Just existing equipment on the shelves today can take months to test and enjoy.

We expanded!

Early on, we discovered we were not able to show the graphic Amiga in our old standard 64 pages, so we have expanded to a new size of 100 pages total. Yet, even at 100 pages, we barely had enough room for some of the better graphic programs and listings.

Graphic programs and reviews, took up the rest of the space and we were not able to show some of the new hardware available.

Kurta Corp. has a line of digitizing products that they produced and used in the IBM and Macintosh markets. In the early stages of the search for products on the Amiga, we contacted them and received a Series One™ Tablet and a Penmouse. Unfortunately, the software drivers for the Amiga took several months longer then they had at first thought.

When the new drivers did come and all the bugs were worked out, we were thrilled with the quality of the Series One™ tablet. We were drawing maps and diagrams and looking continually for an application in the magazine.

I bring up the Kurta tablet to show you a great product that is not reviewed in this issue. Why? We ran out of room. We will move its review as well as a few others back to our next issue.

The Task

So, when we were looking at the cover, we were hit by a double problem. Not only are we looking at four color work, but we should demonstrate some of the capabilities of the Amiga products.

We did!

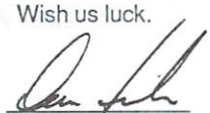
The cover picture (after several arguments over content) was produce with Digi-View from NewTek in Topeka Kansas. We placed a group of the graphic programs available on the table, with a few of our other "art tools" and took the picture in the required three passes. The hardest part was focusing the camera and getting enough light on the subject.

Once we had saved the picture, we tried a hardcopy printout, but we were not able to maintain the brightness of the monitor display.

Our next choice was ImageSet. ImageSet of San Francisco advertises a service that will produce a 35mm slide, poster, or even a four color separation of your Amiga IFF file straight from your disk or via modem.

We opted for a Federal Express delivery rather than a modem transfer, (the Federal Express driver was here before we had the package sealed). We shipped the package on Friday, ImageSet confirmed receipt on Monday and we received the final product on the next Monday.

The picture was as bright as the original screen and should be reproduced on our cover. With a little masking and text additions our masterpiece was ready. Yet, as I write these words, the art is on the table beside me, and we still do not know how the final object will appear. Wish us luck.



Don Hicks

Managing Editor

Dear Sirs:
The story of Amazing Computing™ is amazing to me as revealed in your response to a letter. Thank you for your efforts! I hope the god of entrepreneurs blesses you well and soon enough to avert burnout from a sacrificial level of devotion.

The scope and quality of Amazing Computing™ and the service behind it is great. I am especially glad to see that Amiga™ notes by Rick Rae will be a regular feature. I would like very much to make music with the Amiga™ and am not yet even sure what kinds of things are needed to get done and what they're called.

Persevere, live long and prosper.

Thank you,
David Walton
Ocala, FL

Thanks for the kind words

Dear Amazing Computing™,
Recently, while on vacation, I stopped at a computer store in Minot, North Dakota and discovered your magazine. They had the first five issues and I bought all five. Thanks.

Sincerely,
Jim Williams
Coralville, Iowa

Dear Rick (Wirch),
I was pleased to see your comment in the Marauder review concerning software that requires you to reboot off their disk to use it. I have been making the same point locally and everyone thinks I'm crazy. I hope as the Amiga™ becomes more established and multitasking more commonplace that software companies will get the point.

I've spent quite a bit of time with various software packages figuring out how to use them without rebooting. Deluxe Print™ was especially complicated. By the way, if you haven't gotten around to it, you can put Marauder™ in any directory on any disk and run it if you (1) move its auxiliary files, too; (2) CD to its directory before running it; and (3) (the sneaky part) move a font called HexFont into your fonts library. Marauder™ uses this font for the little display of tracks in the lower right of the screen and freaks if it doesn't find it on the SYS: disk.

As you probably know by now, Marauder™ doesn't work with programs that achieve copy protecting by (ugh) changing disk speed, such as Deluxe Print™. If you call their hotline they tell you how to take the disk apart and twiddle a pot -- not a very elegant solution.

Nathan Walpow
Los Angeles, CA

Dear Amazing Computing™,
I'm writing to thank you for publishing Mr Viveiros' article on building an IBM drive connection which appeared in the May issue of your magazine. I would also like to tell you about an Amazing thing that happened when the project was completed.

It started out as a relatively uneventful project. All of the components were easy to find. I did try five electronic parts houses in an attempt to find the elusive 23 pin D connector. Everyone kept correcting me saying, "You mean 25 pin don't you". I decided that you can't really buy "Anything" in New York. The parts I got matched the recommended ones exactly with two exceptions. One was the fact that the 74LS74 was a 74LS74A. The A extension makes no difference in the operation of the chip so I was not concerned. The second difference was that I had gotten a Shugart SA465 3AA 5.25 drive instead of the recommended SA 455. I thought, "Well it should work". Well it didn't The Amiga wouldn't even boot up with the drive hooked up. I doubled checked all of the connections and everything was perfect. The next thing I did was to play around with the jumper settings but that did no good at first either. Being at my wit's end it was time to get out the documentation that came with the drive. There was a single sheet of paper that said if this drive is installed as drive B, remove the terminater resistor. I had no idea what that was but there was a funny looking chip in a socket on the board so I removed it and continued my game of trying various jumper positions.

I got it to work! Now that was good news but what I got and what I was expecting were two very different things. I was expecting a 5.25 drive that would work as a 360K IBM drive under the transformer and would also configure as a 440K drive under Amiga DOS 1.2. What I got was a 360K IBM drive under the transformer that would also configure as a 720K IBM drive under MS_DOS 3.2. That was exciting but this was Amazing. My new drive auto configures, with absolutely no help from me, as an 880K Amiga drive under both Amiga DOS 1.1 and 1.2 Beta. The disk icon comes up on the workbench screen and everything. The operating systems seems to think it's a 3.5 inch drive. The only minor annoyance is that the system doesn't recognize disk changes in DF2: but DOS 1.2 Beta contains a new command called Diskchange which allows disk swapping in the 5.25 drive. The Commodore 5.25 needs the Diskchange command also but it will only format to 440K and can only be used as an Amiga drive under DOS 1.2. To prove that I really had 880K available I used Diskcopy to copy a Dpaint slide show disk which was 95% full from DFO: to DF2:. Then I started the show and every picture displayed perfectly. Believe It Or Not! Thanks -AC-

Joe Rothman
Sysop of A.M.U.G. BBS
for the Mighty Amiga
New York

Dear Mr. Hicks,
I wish to discuss the subject of standards for Amiga software. There is appearing on the market an increasing number of software products for the Amiga™, and many appear to be simply ported over from who knows where. I recently, purchased by mail a copy of Harvsoft's Infobase. If functions basically as advertised, but it is definitely not something that takes advantage of the Amiga's capabilities. It's basically a simple file handling package written in AmigaBasic. All display is done in black and white. It makes no use of the mouse,

multitasking, multiple windows, softkeys, sounds, animation, color or number crunching; and it's painfully slow slow to operate. Also database names are limited to only six characters in length.

However, Harvsoft shouldn't despair. It's really up to us, the Amiga™ users to specify minimum standards of acceptability for software written for the Amazing Amiga™. Here are some features that should be available on all Amiga software.

1. Color, and lots of it. Allow the user to adjust color displays to suit individual tastes. Remember that a fair number of people suffer from some form of color perception deficiency and may not be able to differentiate between certain hues.

2. Multitasking should be available on word processors, spreadsheets, paint programs, databases etc. This is one of the reasons people buy the Amiga™, so why have software that can't live up to the machine's capabilities.

3. The option of using either pull down menus or keyboard commands. Again, Amiga™ owners want the flexibility of their machine to be reflected in it's software.

4. Commercial software should be supplied in compiled code, for stand alone applications. Who wants to have their disks cluttered up with interpreters? It would be nice if all ads for software state whether or not the application is compiled.

In order to ensure that we the users can locate quality software that makes full use of the machine's awesome power and flexibility, I suggest that a set of Amiga User Acceptability Standards be defined. This could consist of nothing more than a checklist of features that users are required or desirable on various categories of software.

This raises the question of who will develop such a standard? Will Amazing Computing™ do it? You have the required access to Amiga™owners, as well as some very talented writers. I hope you will consider taking on such a survey/project. The standard when completed would be a valuable resource when making objective software comparisons.

Sincerely,
Charles Nielsen
Sydney, N.S.
CANADA

The questions of standards, while at first seems apropriate, can lead to a stale view of the product's versatility. The MS Dos community continualy chants "standards" while the Amiga community has answered with innovation. Where we agree with you in the spirit that all software should perform with some degree of professionalism, there must be room to manuver and develope new means for old ends.

However, AC is only one voice and we welcome our other readers comments and suggestions.

•AC•

PUT YOUR AMIGA TO WORK

with

DATAMAT™

FULLY RELATIONAL DATABASE MANAGEMENT SYSTEM.

- **Now with images in IFF format, display with text/data/voice**
- **Quickly build applications without any program coding from simple phone/mailing list to research to organization-wide information management**
- **Self-running tutorials created automatically for personnel training**
- **Integrate with virtually all existing hardware systems Companion software with identical user-interface for MS DOS, XENIX, UNIX, VMS, and others available. Same application fits all hardware**

DATAMAT PARTIAL SPECIFICATIONS

Organization	Fully Menu-driven Relational Database Management System/Application Generator.	Number of data files per data base	Unlimited
Number of characters per field	1,024	Data types	13 includes Image in IFF Format
Number of fields per record	2,000	Global (System) Fields	40 user definable 9 special purpose
Number of characters per record	4,000	Field checks	Mandatory, Type, Initial value, Value within a specified range.
Number of records per file	4.3 billion		
Multiple response	Supports multiple responses (up to an array of nine) for a single field.	Password security	Field and data base levels
		Calculation capabilities	Full complement of 23 math and trigonometric functions and 13 logical operators. Automatic date and time calculations.
Number of Relations per data file (simultaneous R/W access)	10		

Data Entry - single entry to multiple files and records. **Import/Export** facility with data conversion/reorganization. **Forms Definition** - full screen editor with mini word processor. **Report Generation** - up to 66 lines x 132 columns, 6 level totaling with built in summary. **Sort/Search** - up to 26 selection criteria per query. **Mass Editing, Time Saver Audit** - stores all key strokes used in building application for automatic recreation. **Statistics and Graphics**- stepwise multiple regression, standard statistical tests and analysis; scatter plots, bar charts. Custom Applications Generator - batch/partial batch processing; user-defined menues; self-running demos.

Available through your Amiga dealers. Inquiries Welcome.

Dealers Contact: EJ Distributors (716) 876 1457
3170 Delaware Ave.
Kenmore, NY 14217

Transtime Technologies Corporation

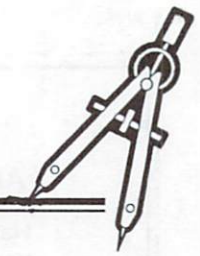
797 Sheridan Drive, Tonawanda, New York 14150; Phone: (716) 874-2010

Datamat is a trademark of Transtime Technologies Corporation
AMIGA is a trademark of Commodore Amiga, Incorporated
MS-DOS & XENIX are trademarks of Microsoft Corporation
UNIX is a trademark of Bell Laboratories
VMS is a trademark of Digital Equipment Corporation

AEGIS DRAW: CAD COMES TO THE AMIGA

A Review

By Kelly G. Adams
Compuserve [71310,226]



Almost everyone has at least one graphics program. Whether it is Electronic Arts' Deluxe Paint or Aegis Images by Aegis Development, or some other package I'm not aware of, these packages are sophisticated and provide the casual or even fairly advanced user with tremendous power.

What does a CAD package give the user to justify its added cost? First off, the user must understand the difference between bit-mapped graphics and object-oriented graphics. The simpler packages mentioned above all rely on bit mapped images: the program itself has no or little concept of things like circles, squares, or even lines.

Once something is on the screen, it is treated as a bunch of individual dots. The dots that make up the circle are exactly the same as the dots that make up the square overlapping it as far as Deluxe Paint is concerned, and so the program can't separate the two images.

Aegis Draw, from Aegis Development, is an object-oriented program. It treats each thing you draw as an individual entity. It stores something like a circle as a center point and a radius, then uses a mathematical process to draw it as what we recognize to be a circle.

With Aegis Draw, I could grab the circle or the square and move it somewhere else without affecting the other, even though they are sitting "on top of" each other.

Another difference between object and bit mapped graphics is that, because an object-oriented system treats each object as a mathematical expression, those objects can be plotted with arbitrarily fine precision.

Essentially, this means that when you output your work on a device of higher resolution than the screen - a plotter, for example - the result will be as perfect as the device in question allows. Circles will be as circular as possible, diagonal lines will be as un-jaggy as can be. In general you will get the best possible resolution.

Also, because an object-oriented package treats graphics elements as mathematical expressions, you can zoom in on a particular point and add ridiculous amounts of detail.

This means it is within reason to have something like a picture of the solar system, and be able to zoom in on the headline of a newspaper held by a properly-scaled graphic image of a person on the surface of the planet Earth!

All this power comes at a price, however: generally, object-oriented graphics packages are slower than bit mapped programs. Everytime something is displayed, thousands of calculations may take place.

Aegis Draw: What Can It DO?

In my personal rating system, three basic factors are considered on a scale of one to 10, plus my own "fudge score" (from one to five stars) which expresses my personal opinion totally uncoloured by factual analysis:

SAF: State of the Art Factor, which essentially indicates to what degree the program takes advantage of the Amiga's power as compared to other similar packages. It also indicates how "innovative" the program is. This is NOT an indication of "goodness" in a program. It is possible for a package to be "State of the Art" without having much substance, in my book, at least. I'd give this a 7.

Speed: Fairly obvious, although this is more a general feeling than a benchmark of performance. I'd give this a 5.

Performance: how well the program carries out the task it was designed for. For example, a graphics package with no print facility would not be "performing" very well unless it was never designed for output and had no need of such. I'd give this a 7.

Adams Score: my star rating; strictly personal opinion, but where it differs significantly from what the other scores indicate I'll usually try to explain myself. Three stars.

Product: Aegis Draw Version 1.0 (May 1986)

Manufacturer: Aegis Development

Notes: Requires 512k RAM

Aegis Draw is a powerful but slightly flawed entry level CAD product with at least one or two rather innovative features. It is a workbench application, which means other programs can run at the same time, within memory limitations.

Compared to existing Draw-type packages on the market such as Mac Draft for the Macintosh, by Innovative Data Design, Inc., and In*A*Vision for the IBM, by Micrographix, PC, Aegis Draw stacks up quite admirably.

A point-by-point comparison is out of the question here, but I'll list the tools available in Aegis Draw and a few of the "preferences" that can be set. Where applicable (ie: when I feel in the mood), I'll draw a few comparisons between Aegis Draw and the other machines' equivalents.

Drawing Tools: "Hand me that Protractor..."

First off, a few points about good design: the guys at Aegis did a nice job in allowing the user to change his mind. To select a tool, you use the familiar menu button to show your choices and select from the pull down menus.

Once you start doing something, like drawing a line or box,

you press the selection button to indicate a starting point, such as the center of a circle, then move the mouse until you have the desired shape (for example, the circle reaches the correct radius) then a second press will complete the operation.

Pressing the menu button at any time during such an operation cancels it. There is an "UnDo" function under the Edit menu for when you make larger mistakes than using the wrong tool.

Now onward to the toolbox: under the tools menu there are two columns of functions. The leftmost column consists of the actual tools themselves, and the rightmost column contains tools for manipulating objects once you've drawn them. A list follows, with descriptions where necessary:

Tools	Description

LINE

RECTANGLE

POLYGON - This allows you to create a multi-sided shape by stretching a set of lines between points chosen one at a time with the left mouse button.

FREEHAND - Unlike a bit-mapped graphics system, freehand drawing is the least used function. It uses a lot of memory (it is stored, I believe, as zillions of line segments of arbitrarily small length).

ARC - In Aegis Draw, the user selects the center of the arc, and the beginning and end points. Other packages, such as Pro Design II by American Small Business Computers for the IBM PC, allow a number of different options for such an arc. All Mac Draw-type packages I know don't allow such diversity, but then, of course, they are easier to use....

CIRCLE

TEXT - This is not normal Amiga font type text. You don't have any choice of type style. The text in Aegis Draw is vector-drawn, which is another way of saying that it is stored mathematically by the program. This translates into the user being able to size and rotate the letters to his hearts' content without loss of resolution.

DIMENSION - The dimension tool causes Aegis Draw to automatically calculate and display the distance between two points. This is selected as if you were drawing a line. The distance in units is actually entered into the diagram in between two arrows that follow the line between the beginning and end points chosen.

PART - A tremendously powerful feature of Aegis Draw is its ability to maintain a database of regularly used parts. An example of such an object might be a piece of furniture for an architectural diagram or an electronic component for a schematic.

The Part command lets you insert a part you defined and named using the Edit menu Group function: the power of the parts list is shown, you use the same parts list with several drawings.

Other packages have similar features. In*A*Vision on the IBM has"templates" - essentially a second diagram window. Aegis also supports this, but this parts concept is quite rare in software of its size.

Aegis has promised to extend this function in the soon-to-be-released Aegis Pro Draw. The database will include attributes, such as part cost.

MANIPULATORS

DRAG IT - This moves an object from place to place.

ROTATE - The selected object can be rotated around a user-defined point. Control of rotation can be very precise. In comparison, *A*Vision only allows 90 degree rotations.

CLONE

ERASER

EXPLODE - For those not accustomed to object-oriented graphics systems, "Explode" may sound a little strange. Normally, an object is any one thing drawn with a tool. However, a new object can be formed from a bunch of objects. The explode manipulator reverses this, discombobulating a single object made of a number of sub-objects.

SIZER

BACK - Because Aegis Draw considers objects as individual constructs, it is possible for one item to get "in front of" another. The back manipulator moves a selected object behind another, thus giving a choice of the drawing order.

COLOR - This does more than just allow the change of an objects color. Aegis Draw allows objects to be different colors. It also sets the line thickness and fill pattern of an object.

STATS - If you require that a circle should be centered at (220,327), this command will allow you to enter those numbers directly. Exactly what you can change depends on the type of object you are manipulating: text, for example, allows you to change location, height and width, and the text itself.

Preferences...

Besides the actual objects you can create and manipulate using the tools in Aegis Draw, there are also a number of user-definable options. For example:

RULERS - It is possible to turn these on or off, or to select the unit type, Metric or Imperial.

GRID SNAP - When this option is turned on, all drawing operations are forced to start and end at a grid boundary.

SMOOTHING - This causes straight line objects to be smoothed. For example, a sequence of saw-tooth lines becomes a sine wave.

Aegis Draw has a multiple layer capability, up to 250 layers per diagram, that allows the user to make certain layers invisible and to select a particular layer to work on.

This is quite useful for maintaining a number of revisions to an item. You can regress to a particular revision number by turning layers off, or to show and allow editing to only certain aspects of a project, while protecting others.

The package also supports multiple diagrams in memory at once. With 512k you are limited to about two. Any other programs running in the background would limit this further.

Finally, Aegis Draw is capable of producing IFF standard picture files. You can edit pictures created with Draw in Deluxe Paint, Aegis Images, or any other IFF standard program that comes along.

Of course, there are a zillion other options and features: this package is a tad too inclusive for anything much shorter than a book to fully describe.

Aegis' standard procedure of producing what looks like a third party book for each of their products has worked again (wonderfully here, by the way). It is, however, not perfect...

Nothing is Perfect

My major complaints with Aegis Draw 1.0 center on its output options.

First, let me commend Aegis for developing an intelligent plotter driver system which allows the user to write their own drivers. With this system available, it seems almost criminal to include only one pre-made driver, for the Roland DXY-980.

This driver supports a graphics plot standard called HPGL (Hewlett Packard Graphics Language, I believe), but even so it seems almost mean-hearted not to do a little leg work for the user and include completed drivers for a few more plotters.

Even worse, in my opinion, is that no reasonable option exists for high-quality printer (not plotter) output.

What printer output exists is essentially a screen dump, showing things like the menu bars. This results in the typical appalling resolution one normally gets from a bit-mapped graphics package. The average user may not have access to high-power equipment. I would love to have printer output quality comparable to that of Pro Design II.

As a final complaint, the manual claims there is an option to adjust plotting height and width, but the addendum card says correctly that this option was removed.

On the card, Aegis claims the ability to select the output area was considered redundant because the information was already stored in the plotter driver.

What this means is that if I want to create an 8.5 x 11" plot on

my 11 x 17" plotter I must re- write the plotter driver!! Preposterous!

Finally, and this is a minor complaint, Aegis Draw could be faster. MacDraft is substantially quicker, although I performed no intensive benchmarks: there is no real reason for this.

It all Ends When Its Done...
All in all, this is a very good package for the serious graphics user who owns a plotter. Until and if Aegis brings out real printer drivers to produce the quality of output that Aegis Draw obviously is capable of, it is difficult to recommend this package to anyone without a plotter of their own.

The price is reasonable, and the manual mentions an upgrade path to ProDraw, once it is available, for the difference in price between the two.

Also, Draw is not copy protected. A practical, professional quality application like this must be backed up. Aegis Draw certainly deserves to do well - especially if they attend to my complaints!

Last Minute News...
As I was getting ready to send this off to Amazing Computing, I got a phone call from Aegis. They told me of a new version of Aegis Draw, which should be available by mid-August. Keeping in mind that I have not seen this package, here is a list of what version 1.10 will supposedly include:

A "Pick Plotter" requester, that will allow you to choose from a number of plotter drivers in a drawer on the system disk. Aegis will also include about five new drivers in this drawer. This addresses my complaint about a lack of plotter drivers.

Feet, inches, fractions units option, for dimension and ruler lines. This allows increments as small as 1/128 of an inch.

Improved numeric display, showing angles of rotation, height and width of rectangles, length of lines, etc.

Summary
All in all, Aegis seems to have covered most of my complaints: they also fixed a few bugs, including one which I never noticed, I am embarassed to say. Apparently, the "smooth" preference formerly turned straight lines into wavy lines....

They still have to add high quality printer drivers, but the person I talked to, before I even mentioned that this was one of my major complaints, told me that they were developing better "printer plot" routines. Here, here, guys!

Also, Aegis has a low or no cost upgrade policy: contact them for details.

I was also asked to mention that approximately ninety percent of these upgrades were based on comments from users. It seems Aegis really wants to hear from their clients. Anyway, if version 1.10 is as good as it sounds, then it should be pretty great.

•AC•

TRY 3D!

An introduction to 3D graphics

By **Jim Meadows**
Compuserve [75046,2012]

Shortly after getting my Amiga, I was interested in testing the speed of Amiga Basic. I decided to convert some three-dimensional routines from the IBM PC and see how fast they would run. (These routines were on my Apple before that.)

I was excited to find that Amiga Basic was fast enough to see results of moving and rotating simple 3D images. To get the speed needed to do this, it required assembler on the Apple and compiled Basic on the IBM PC.

I realized that the Amiga would be an excellent tool to introduce others to the world of 3D graphics. So I added menu and mouse control and developed the sample program that appears with this article. I also decided to write this article so others could learn the basics of 3D graphics.

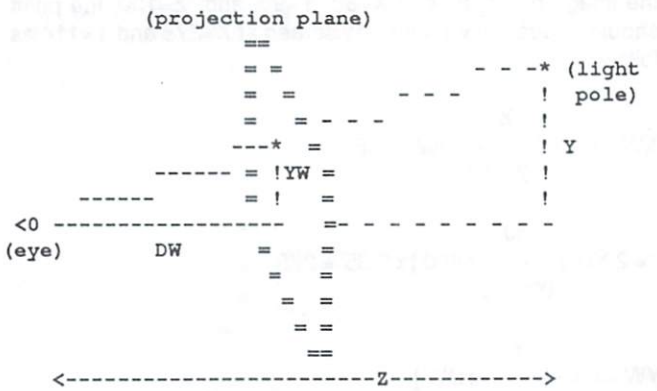
Let us begin by asking, 'How can you draw a 3D image on a two-dimensional display screen?' The first concept we must master is that of a projection plane.

Imagine that you are looking out of a window at a light pole outside. If you reached up and marked on the window the top and bottom of the light pole as you saw it, and then connected the two points you marked on the window, you would then have projected the light pole image onto a projection plane -- the window.

The size of the light pole that you drew on the window depends on 3 things: how tall the light pole is, how far away it is, and how far away your eye is from the window.

To represent a point in 3D space, you can use X, Y, and Z values where X is horizontal distance, Y is vertical distance, and Z is depth (or how far away it is). The light pole, window, and your eye are shown in Figure 1 using these designations.

FIGURE 1



If the telephone pole is Y units high, the image you draw on your window will be some amount we will call YW units high. We will call the distance from your eye to the window DW units.

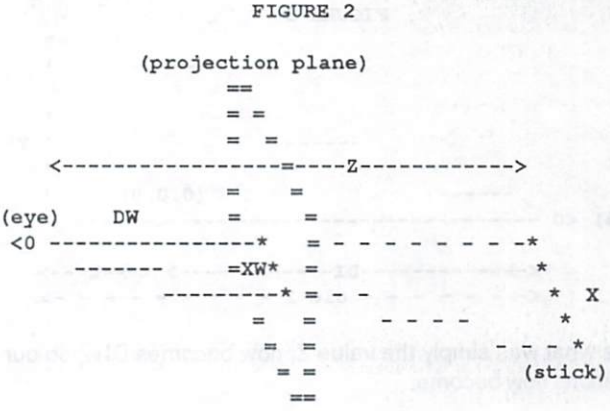
By looking at Figure 1 you can see two triangles: a big one formed from your eye to the light pole, and a smaller one formed from your eye to the light pole image you drew on the window. Since these two triangles are similar triangles, the ratio of their sides will be the same. This allows us to write:

$$\frac{YW}{DW} = \frac{Y}{Z}$$ and solve for YW:

$$YW = \frac{Y}{Z} \times DW$$

If you looked at a stick lying on the ground and drew its image on the window, the same thing applies in the horizontal direction as shown in Figure 2.

FIGURE 2



If the stick is X units long and the image you draw on the window is XW units long, we again see two triangles and can write:

$$\frac{XW}{DW} = \frac{X}{Z}$$ and solve for XW:

$$XW = \frac{X}{Z} \times DW$$

If you were looking at a box outside on the ground and you marked the corners on your window and connected the

points with lines, you would then have a 3D image projected onto your window, just as you did with the light pole and stick.

Now think of the window as your display screen. By defining an image as a series of points in 3D space connected by lines, you can use the XW and YW equations to compute the X & Y values on your display screen that correspond to each of the points of the image. If you connect the points computed on your display screen, you will have a 3D image on your display.

Before we can consider how to rotate a 3D image so that we can 'view' it from different angles, we need to adjust the XW and YW equations a little bit more.

It is desirable for an object to appear to 'spin' about a point at the center of the object when it is rotated. It is easier to develop the rotation equations if the center of the object is at the point (0,0,0).

In the diagrams used so far, we assumed our eye was at (0,0,0) and the object was defined at some distance Z from our eye. If we define an image centered around the point (0,0,0) and our eye is also at (0,0,0), then it would appear as if we were inside the object!

To remedy this situation, we need to move the point (0,0,0) from where our eye is to the location that we want the image to appear to rotate around. If we choose a variable DI to be the Distance to the Image rotation point, then we represent the larger triangle as shown in Figure 3.

```
                         FIGURE 3
                                                  ------*
                                           ------       !

                                    ------              !
                               ------                   ! Y
                          ------                        !
                     ------              (0,0,0)        !
(eye) <0 -------------------------------------O---------

         <-------------DI-------------> <---Z--->
         <- - - - - - - old Z - - - - - - - - - ->
```

Thus what was simply the value Z, now becomes DI+Z so our equations now become:

$$XW = \frac{X}{Z+DI} \times DW \quad \text{and}$$

$$YW = \frac{Y}{Z+DI} \times DW$$

Since DI is the distance from our eye to the 'center' of the image, the above equations also allow us to vary how 'far' the image appears to be from us by varying DI. By using the equations for YW and XW, any point in 3D space could be projected onto your two dimensional window as shown in Figure 4.

```
                    FIGURE 4

       (projection plane)                 +
               ==                       Y .
               = =                        .
               =  =                       .
               =   =               - - - - - - *
           (XW,YW) = - - - -              .(X,Y,Z)
         ----------* = -                  .
    ---------      =     =                .
<0 ------------------    =- - - - O - - - +.
  <=========================DI==> <==Z==>  .
  <-------DW------->     =                  .
                      =  =                   .
                       = =                 X  +
                        ==
```

We need just two more adjustments to these equations for use on the Amiga. Due to the difference in horizontal and vertical screen resolution, a scaling factor should be included in order to make a square actually look 'square'.

To accomplish this, the computed values of X should be multiplied by a screen scaling factor SF. Because Y increases as you move down on the Amiga screen, we should reverse the sign of the computed Y value. Finally, so that we can move the image around on the screen, we add PX and PY values to the computed X and Y values. So we then have:

$$XW = PX + \left(\frac{X}{Z+DI} \times DW\right) \times SF$$

$$YW = PY - \left(\frac{Y}{Z+DI} \times DW\right)$$

You will find the above equations used in the accompanying program.

For the Amiga I have found the scaling factor for the high resolution display should be SF = 2.35, and a viewing distance of DI = 900 units with the projection plane at DW = 400 units works well for the images defined in the program. (If you change to a different screen resolution, adjust SF accordingly.)

As an example, let's say you want an image positioned at PX=200 and PY=150 on your screen. If one of the points of the image had values of X=80, Y=90, and Z=100, the point should appear on your display screen at X=275 and Y=114 as follows:

$$XW = PX + \left(\frac{X}{Z+DI} \times DW\right) \times SF$$

$$= 200 + \left(\frac{80}{100+900} \times 400\right) \times 2.35 = 275$$

$$YW = PY - \left(\frac{Y}{Z+DI} \times DW\right)$$

AMIGA HAS MULTI-TASKING, DISCOVERY SOFTWARE USES IT!

FROM NOW ON YOU CAN PRINT OR SAVE ANY SCREEN, FROM ANY PROGRAM, ANY TIME!

GRABBiT takes WYSIWYG* to the limit. With **GRABBiT** you capture exactly what you see on your screen in an instant, regardless of what other programs you're running. **GRABBiT** works with all AMIGA video modes, including **"Hold-and-Modify"**. It even lets you capture images from animated programs, like the bouncing ball in Boing! What's more, **GRABBiT runs completely in the background** – transparent to your other software. **GRABBiT** is **always** ready for you to use, even while you're in the middle of another program. As if that's not enough, **GRABBiT** requires only about 10K of your precious RAM to operate, and it supports dozens of printers. It's not a game, it's not a toy, **GRABBiT** is truly a productivity power tool for your AMIGA!

We believe powerful software should be easy to use. **GRABBiT is one of the EASIEST programs you'll ever use!** Every **GRABBiT** operation is triggered by one of the "HotKeys", a set of easy-to-remember key sequences that only take minutes to learn. Each HotKey is generated simply by holding down the "**Control**" and "**Alt**" keys and pressing one of the designated letter keys. What could be easier?

You won't grow old waiting for **GRABBiT** to finish printing, either. When we say multi-tasking, we mean it. **GRABBiT** has **a unique TPM (Task Priority Monitor) module** which makes sure your other software can still run even while **GRABBiT** is printing. The TPM module constantly tracks **GRABBiT's** printing priority, making sure it is neither too high nor too low, **but always just right! GRABBiT** adds a new dimension to the AMIGA's multi-tasking capability.

GRABBiT supports **dozens of different printers** because it uses the standard Amiga device drivers. Any printer you can choose in "Preferences" is **automatically** supported by **GRABBiT**. You'll get the most from color printers too, because **GRABBiT** supports full-color printing. In fact, we have seen amazing color printouts produced by **GRABBiT** on the Oki-Mate 20, a color printer costing less than $200.00.

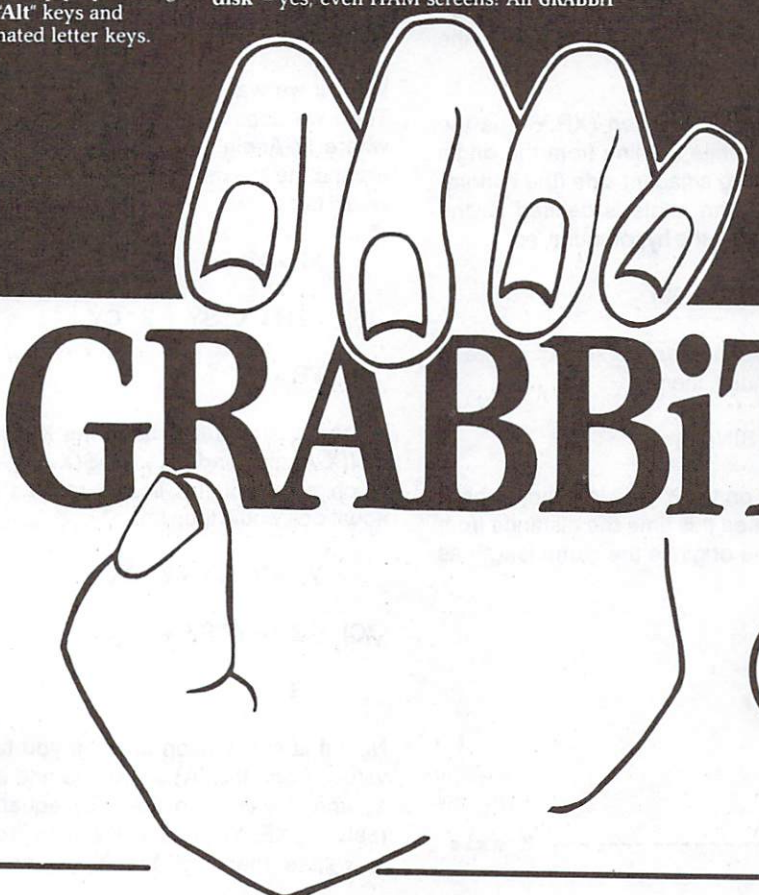
Of course, **GRABBiT's** abilities are not limited **merely** to printing; **GRABBiT** is equally adept at **saving screen images to disk** – yes, even HAM screens! All **GRABBiT** disk files are saved in the popular **IFF format**, the emerging graphics standard for AMIGA. You can capture any screen to disk for slide-show presentations or later enhancement with any popular AMIGA graphics editor like AEGIS Images or Deluxe Paint. We even include a specially modified PD utility called "SEE", which allows you to view IFF image files quickly and easily. **GRABBiT's** disk operations are **lightning fast** because **GRABBiT** is written in a hybrid of highly optimized C and 68000 Assembler.

Once you start using **GRABBiT you'll want it on every disk**. You can easily install **GRABBiT** in your system startup-sequence, so it will **always be there when you need it**. With all its features this would be a great package at any price. But we think you'll agree with us that **GRABBiT's** most outstanding feature is **VALUE!** You get all the power of this sizzling new software for an unbelievably low

$29.95

+ $5 Shipping & Handling

VISA MasterCard



GRABBiT

*WHAT YOU SEE IS WHAT YOU GET

Amiga is a registered trademark of Commodore Business Machines

Images is a registered trademark of AEGIS Development Corp.

Deluxe Paint is a registered trademark of Electronic Arts.

Oki-Mate 20 is a registered trademark of Okidata of America.

DISCOVERY SOFTWARE INTERNATIONAL

262 South 15th Street
Suite 400
Philadelphia, PA 19102

(215) 546-1533

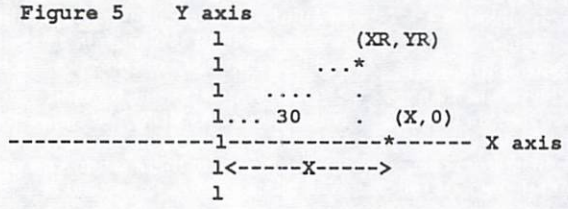
$$= 150 - \left(\frac{90}{100 + 900} \times 400\right) \quad = 114$$

Now let's consider how to rotate an image for viewing from different angles. (If you want to skip the math and go to the results, skip down to the paragraph that begins 'The resulting equations then become...').

Imagine a line going straight up and down from the center of your image and call it the Y axis. A line going from the center of the image out to the right and left will be the X axis.

The Z axis would be a line from your eye through the center of the image. Picture a point to the right on the X axis and then rotate that point 30 degrees around the place where the X and Y axis intersect. Where will the new location of that point be?

Figure 5



As shown in Figure 5, the new location of the point has a smaller X value, XR. Also, the point is no longer on the X axis, so Y is no longer 0, but has a positive value, YR.

Since the distance from the new location to the origin (where the X and Y axis cross) remains the same during a rotation, the line from the origin to the new (XR,YR) location is the same length as the original X value.

The line from the origin to the new location (XR,YR) is the hypotenuse of a right triangle, while the line from the origin to the point XR on the X axis is the adjacent side (the bottom) of the triangle. The COSINE of an angle is defined as the side adjacent to the angle, divided by the hypotenuse, so:

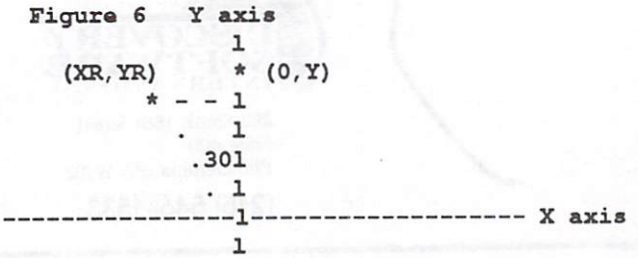
COS(30) = XR/X or XR = X * COS(30)

Since the SINE of an angle is defined as the side opposite to the angle divided by the hypotenuse, then:

SIN(30) = YR/X or YR = X * SIN(30)

What if the point was originally on the Y axis instead of the X axis? Looking at Figure 6, we see this time the distance from the new (XR,YR) location to the origin is the same length as the original Y value.

Figure 6



The new X value (XR) in this case is negative and could be found by:

SIN(30) = -XR/Y or XR = -Y * SIN(30)

The new Y value (YR) could be found by:

COS(30) = YR/Y or YR = Y * COS(30)

By combining the point-on-the-X-axis and the point-on-the-Y-axis cases we get:

XR = X * COS(30) - Y * SIN(30)

YR = X * SIN(30) + Y * COS(30)

ZR = Z

The above equations can be applied to any point in order to rotate it 30 degrees around the Z axis. The ZR equation was included to show that since the point rotates about the Z axis, the Z value of the point would not change.

If we use a variable SZ = SIN(Z-Angle) and CZ = COS(Z-Angle) where 'Z-Angle' is how much we want the image rotated around the Z axis, then the equations become:

XR = X * CZ - Y * SZ

(A) YR = X * SZ + Y * CZ

ZR = Z

What if we want to rotate a point around the Y axis instead? Then we use SY = SIN(Y-Angle) and CY = COS(Y-Angle) where 'Y-Angle' is how much we want the image rotated around the Y axis, and by similar development the equations would be:

XR = X * CY - Z * SY

(B) ZR = X * SY + Z * CY

YR = Y

What if it is rotated around the X axis? Again by using SX = SIN(X-Angle) and CX = COS(X-Angle) where 'X-Angle' is how much we want the image rotated around the X axis, the equations would then be:

YR = Y * CX - Z * SX

(C) ZR = Y * SX + Z * CX

XR = X

Now it is substitution time. If you take the XR, YR, and ZR values from the (A) equations and substitute them for the X, Y, and Z values in the (B) equations, and then take the resulting XR, YR, and ZR values from the (B) equations and substitute them for the X, Y, and Z values in the (C)

equations, you will get a general set of equations for simultaneous rotations around any of the Axes.

(Instead of substituting, you could convert the (A), (B), and (C) equations into matrix form and multiply them together to get the same results).

The resulting equations then become:

```
XR = X * (CY*CZ)
   + Y * (-CY*SZ)
   + Z * (-SY)

YR = X * (CX*SZ-SX*SY*CZ)
   + Y * (CX*CZ+SX*SY*SZ)
   + Z * (-SX*CY)

ZR = X * (SX*SZ+CX*SY*CZ)
   + Y * (SX*CZ-CX*SY*SZ)
   + Z * (CX*CY)
```

These are the equations used in the accompanying 3D program to rotate all the points of an image before projecting them onto the projection plane.

You will notice that the values in parentheses in the above equations can be evaluated once, and then applied to all the points in the image table, array IT%, in order to effectively rotate the image in space.

The computed XR, YR, and ZR values are stored in a rotated image table, called RIM%. Finally, the projection equations developed earlier are used to 'see' the rotated 3D image on the projection plane (your display screen).

Due to the sequence in which we evaluated the rotation equations, an image may appear to spin about an axis or 'wobble' about an axis when rotated depending on the angles of rotation.

One final comment about the rotation calculations is needed. We normally think of angles in degrees (e.g., 90 degrees is a right angle). However, the SINE and COSINE functions in Basic expect the angles to be in radians.

Since 2 pi radians = 360 degrees, to convert degrees to radians you must divide by 57.2958. This is done in the program as it computes the trig functions.

The accompanying program uses the projection equations and the rotation equations we have developed. When you run the program, it first greets you with a 3D image that rotates at different X, Y, and Z angles and varies the distance the image is from you for different effects.

Then another image is displayed and rotated, using a draw/undraw technique where the image is redrawn using the background color in order to erase it.

Next the image is 'flown' through space by a short routine that moves the image about, rotating it, and moving it 'closer' to your eye by varying DI.

Another image is then 'flown' through the same flight pattern. When the 'flight' is over, the program switches to a manual display mode with menus that allow you to rotate the last image for viewing at any angle and to move it about. Holding the left mouse button down repeats the last menu item selected for continuous movement.

The images displayed are sometimes called 'wire frame' images, since all edges are visible. You appear to see through the object, as if you have X-ray vision. By adding hidden line removal routines, only the surfaces you normally would see would be drawn. However, this type of routine can become quite complex, and is beyond the scope of this introduction to 3D graphics.

The variable EF in this program is used as an 'Erase Flag'. If EF=0, the image is drawn using the colors found in the image table. If EF=1, the image is drawn using the background color, effectively erasing the image from the screen. If EF=-1, the screen will be cleared after the rotated image is computed and before the image is drawn on the screen.

One possible use of these routines would be to combine them with a drawing program. An image could be defined, displayed from various angles, and saved on disk to later be filled in with details using a drawing program. Or for those so inclined, the image table could be expanded and equations used to generate entries in the table for the points of a 3D surface. Then the surfaces could be rotated and viewed from various angles.

The images are defined at the end of the program. Try substituting your own images in place of the program's images. The first value of a table entry is the color to be used to connect the point with the previous point. If 0 is used for the color, a line is not drawn.

The remaining three values are the X, Y, and Z values of the point. To mark the end of your image use -1 for the color of the last entry. When you define the image points, remember that the object will appear to rotate around the point (0,0,0).

Have fun in the 'real' world of 3D graphics.

```
' Try3d -- An Example of 3D Programming
' Initial Amiga implementation
'  by Jim Meadows 6/1/86 Compuserve [75046,2012]

' 3D Greeting
CLS:PRINT
COLOR 3:PRINT "    Try3D";
COLOR 1:PRINT " -- An Example of 3D Programming"
COLOR 1:PRINT "             by Jim Meadows"
GOSUB InitVals
GOSUB SetImage
' GOSUB SetImage
' GOTO Manual
ef=-1
ax=10:ay=5:px=180:py=70
GOSUB DrawImage
LOCATE 18,20:COLOR 3:PRINT "Hello!"
GOSUB Pause
ax=10:ay=-30
```

```
GOSUB DrawImage
LOCATE 18,3:COLOR 3:PRINT "Welcome to the World of
3D Graphics!"
GOSUB Pause
ax=30:ay=20
GOSUB DrawImage
LOCATE 18,15:COLOR 3:PRINT "Hmmmmmm....."
GOSUB Pause
ax=-30:ay=-15:az=20:di=500
GOSUB DrawImage
LOCATE 18,8:COLOR 3:PRINT "Yep, just as I figured
..."
GOSUB Pause
az=-20:ay=15
GOSUB DrawImage
LOCATE 18,4:COLOR 3:PRINT "You've got what it takes
for 3D!"
GOSUB Pause
ax=10:ay=-20:az=0:di=900
GOSUB DrawImage
LOCATE 18,15:COLOR 3:PRINT "You have ..."
GOSUB Pause
ax=-5:ay=0:di=700
GOSUB DrawImage
LOCATE 18,15:COLOR 3:PRINT "... an AMIGA!!"
FOR i = 1 TO 10000:NEXT
ef=0

Ri:
'  Rotating Image
CLS
LOCATE 19,13:COLOR 3:PRINT "Delta Wing Fighter"
GOSUB SetImage
' Draw and Undraw rotating image
ax=-90:ay=270:az=0:px=160:py=100
FOR ii = 1 TO 4
  GOSUB DrawImage
  GOSUB Pause
  ef=1
  GOSUB DrawImage
  ef=0
  ax=-20
  ay=ay-60
  IF ii=1 THEN ax=0:ay=270:az=0
NEXT

Fly:
' Animated flight path
FOR inum = 1 TO 2
  IF inum=2 THEN GOSUB SetImage
  di=2400
  ax=0:ay=270:az=0
  px=30:py=30:ef=-1
  GOSUB DrawImage
  px=px+40:ax=ax-5:di=di-100
  GOSUB DrawImage
  ax=ax-5:di=di-100
  FOR R=1 TO 3
    px=px+40:ay=ay-20:di=di-100
    GOSUB DrawImage
  NEXT
  FOR R = 1 TO 3
    px=px+30:ay=ay-20:az=az-20:di=di-100
    GOSUB DrawImage
  NEXT
  FOR R=1 TO 4
    px=px-20:py=py+10:az=az+10:di=di-80
    GOSUB DrawImage
  NEXT
  FOR R=1 TO 8
    px=px-9*R:py=py+10:az=az+5:di=di-60:ax=ax-5
    GOSUB DrawImage
  NEXT
NEXT

' Finally allow manual control
GOSUB SetImage

Manual:

MENU 1,0,1,"Rotate +"
MENU 1,1,1,"Around X-axis"
MENU 1,2,1,"Around Y-axis"
MENU 1,3,1,"Around Z-axis"

MENU 2,0,1,"Rotate -"
MENU 2,1,1,"Around X-axis"
MENU 2,2,1,"Around Y-axis"
MENU 2,3,1,"Around Z-axis"

MENU 3,0,1,"Move"
MENU 3,1,1,"Closer"
MENU 3,2,1,"Away"
MENU 3,3,1,"Right"
MENU 3,4,1,"Left"
MENU 3,5,1,"Up"
MENU 3,6,1,"Down"

MENU 4,0,1,"Reset"
MENU 4,1,1,"Angles"
MENU 4,2,1,"Distance"
MENU 4,3,1,"Position"
MENU 4,4,1,"Quit"
```

```
ON MENU GOSUB Menus
ON MOUSE GOSUB Mous

m1=1:GOSUB Reeset
m1=2:GOSUB Reeset
m1=3:GOSUB Reeset
act=1:ef=-1

MOUSE ON
MENU ON

Loop:
  IF act=0 THEN inc=1: GOTO Loop
  GOSUB DrawImage
  GOSUB Vals
  IF MOUSE(0)<>-1 THEN act=0 ELSE GOSUB Mous
  GOTO Loop

'-------------
' Subroutines
'-------------
Vals:
  COLOR 1
  LOCATE 1,1:PRINT "Ax,Ay,Az: "ax","ay","az
  LOCATE 2,1:PRINT "Px,Py    : "px","py
  LOCATE 3,1:PRINT "Di       : "di
  COLOR 3
  LOCATE 4,1:PRINT "Use Menus to Change View"
  COLOR 2
  LOCATE 5,1:PRINT "(press left button to repeat)"
RETURN

Menus:
 act=1
 inc=1
 m0=MENU(0)
 m1=MENU(1)
 ON m0 GOSUB RotateP,RotateM,MoveI,Reeset
RETURN

Mous:
  act=1
  inc=inc+.5
  ON m0 GOSUB RotateP,RotateM,MoveI,Reeset
  RETURN

RotateP:
  IF m1=1 THEN ax=ax+10*inc
  IF m1=2 THEN ay=ay+10*inc
  IF m1=3 THEN az=az+10*inc
RETURN

RotateM:
  IF m1=1 THEN ax=ax-10*inc
  IF m1=2 THEN ay=ay-10*inc
  IF m1=3 THEN az=az-10*inc
RETURN

MoveI:
  IF m1=1 THEN di=di-50*inc
  IF m1=2 THEN di=di+50*inc
  IF m1=3 THEN px=px+20*inc
  IF m1=4 THEN px=px-20*inc
  IF m1=5 THEN py=py-10*inc
  IF m1=6 THEN py=py+10*inc
RETURN

Reeset:
  IF m1=1 THEN ax=-15:ay=-25:az=0
  IF m1=2 THEN di=1200
  IF m1=3 THEN px=160:py=100
  IF m1=4 THEN  MENU OFF:END
RETURN

Pause:
  FOR i = 1 TO 4000:NEXT
RETURN

' ---------------------------------
' |          3-D Routines         |
' ---------------------------------

' ax,ay,az = rotation Angle in degrees
' di = distance to image
' dw = distance to window (projection plane)
' px,py = position of image on screen
' sf = screen scaling factor
' ef = erase flag (1=erase, 0=draw, -1=cls & draw)
' Image data is at end of program

InitVals:
  ' Define Arrays
  DIM it%(100,3):' Image Table
  DIM rim%(100,3):' Rotated Image
  ' Initialize Values
  x=0:y=0:z=0
  dw=400:' Distance to window
  di=900:' Distance to image
  sf=2.35:' Screen scale factor
  ax=0:ay=0:az=0:' Angles in Degrees
  px=200:py=100:' x,y Image Location
  ef=0:' Erase Flag
  f=57.29578:' Degrees to Radians factor
RETURN
```

```
DrawImage:
' Draw the Image
    GOSUB Rotate
    GOSUB DrawIt
RETURN

Rotate:
 ' First get trig values from angles
 sx=SIN(ax/f):cx=COS(ax/f)
 sy=SIN(ay/f):cy=COS(ay/f)
 sz=SIN(az/f):cz=COS(az/f)
 ' Then compute rotation values
 xRx=cy*cz
 yRx=-cy*sz
 zRx=-sy
 xRy=cx*sz-sx*sy*cz
 yRy=cx*cz+sx*sy*sz
 zRy=-sx*cy
 xRz=sx*sz+cx*sy*cz
 yRz=sx*cz-cx*sy*sz
 zRz=cx*cy
 ' Now Rotate Image
 np=0
 Rotate1:
  ' Get next point
  c=it%(np,0):IF c=-1 THEN RETURN
  x=it%(np,1):y=it%(np,2):z=it%(np,3)
  ' Compute its new location
  rim%(np,1)=x*xRx+y*yRx+z*zRx
  rim%(np,2)=x*xRy+y*yRy+z*zRy
  rim%(np,3)=x*xRz+y*yRz+z*zRz
  np=np+1
  GOTO Rotate1
```

```
DrawIt:
 np=0:IF ef=-1 THEN CLS
 DrawIt1:
  ' Check for end of table
  c=it%(np,0):IF c=-1 THEN RETURN
  ' Keep from dividing by zero
        IF     (rim%(np,3)+di)     =     0     THEN
rim%(np,3)=rim%(np,3)+1
  ' Compute screen x & y
  xw=px+(rim%(np,1)/(rim%(np,3)+di))*dw*sf
  yw=py-(rim%(np,2)/(rim%(np,3)+di))*dw
  ' Draw next line or move to next point
  IF c=0 THEN GOTO JustMove
  colr=c:IF ef=1 THEN colr=0
  LINE (lx,ly)-(xw,yw),colr
  JustMove: lx=xw:ly=yw
  np=np+1
  GOTO DrawIt1


SetImage:
' Routine to insert an image into the table
  n=0
  itloop:
    READ it%(n,0)
    IF it%(n,0)=-1 THEN RETURN
    READ it%(n,1),it%(n,2),it%(n,3)
    n=n+1:GOTO itloop


' Greeting Image
' Image Data Format:c,x,y,z
'    (c=color, if =0 then move w/o drawing)
  DATA 0,-50,30,0
  DATA 1,-55,35,10
  DATA 1,-45,0,0
  DATA 1,-20,-60,-30
  DATA 1,20,-60,-30
  DATA 1,20,-60,-30
  DATA 1,45,0,0
  DATA 1,55,35,10
  DATA 1,50,30,0
  DATA 3,30,80,-30
  DATA 3,-30,80,-30
  DATA 3,-50,30,0
  DATA 0,0,22,-30
  DATA 1,0,-4,-36
  DATA 0,-5,0,-30
  DATA 1,0,-4,-36
  DATA 1,5,0,-30
  DATA 0,-20,30,-25
  DATA 1,-35,25,-17
  DATA 1,-20,20,-25
  DATA 1,-5,25,-21
  DATA 1,-20,30,-25
  DATA 2,-20,20,-25
  DATA 0,20,30,-25
  DATA 1,35,25,-17
  DATA 1,20,20,-25
  DATA 1,5,25,-21
  DATA 1,20,30,-25
  DATA 2,20,20,-25
  DATA 0,-20,-26,-22
  DATA 3,0,-34,-30
  DATA 3,20,-26,-22
  DATA 0,-10,-30,-26
  DATA 3,10,-30,-26
  DATA -1
```

```
' Delta Wing Fighter Image
  DATA 0,0,-20,100
  DATA 1,0,20,-100
  DATA 0,50,-20,-100
  DATA 1,0,-20,100
  DATA 1,-50,-20,-100
  DATA 2,0,20,-100
  DATA 2,50,-20,-100
  DATA 2,-50,-20,-100
  DATA 0,-75,0,-100
  DATA 3,0,0,0
  DATA 3,75,0,-100
  DATA 3,-75,0,-100
  DATA -1:' End of Image

' Chaser Image
  DATA 0,-25,0,-25
  DATA 1,25,0,-25
  DATA 1,25,0,25
  DATA 1,-25,0,25
  DATA 1,-25,0,-25
  DATA 0,-25,25,25
  DATA 3,-25,-25,25
  DATA 3,-25,-25,-25
  DATA 3,-25,25,-25
  DATA 3,-25,25,25
  DATA 0,25,25,25
  DATA 3,25,-25,25
  DATA 3,25,-25,-25
  DATA 3,25,25,-25
  DATA 3,25,25,25
  DATA 0,0,0,-25
  DATA 2,0,0,50
  DATA 2,0,10,25
  DATA 2,0,0,-25
  DATA -1

' XYZ axis Image
  DATA 0,-100,0,0
  DATA 1,100,0,0
  DATA 1,80,-20,0
  DATA 0,100,0,0
  DATA 1,80,20,0
  DATA 0,140,14,0
  DATA 1,170,-16,0
  DATA 0,140,-16,0
  DATA 1,170,14,0
  DATA 0,0,-100,0
  DATA 2,0,100,0
  DATA 2,20,80,0
  DATA 0,0,100,0
  DATA 2,-20,80,0
  DATA 0,0,120,0
  DATA 2,0,134,0
  DATA 2,14,148,0
  DATA 0,0,134,0
  DATA 2,-14,148,0
  DATA 0,0,0,100
  DATA 3,0,0,-100
  DATA 3,0,-20,-80
  DATA 0,0,0,-100
  DATA 3,0,20,-80
  DATA 0,-14,14,-140
  DATA 3,16,14,-140
  DATA 3,-14,-16,-140
  DATA 3,16,-16,-140
  DATA -1
```

•AC•

Finally!

ASDG, Inc. is Proud To Announce Availability of the First Intelligent Hardware Expansion Path for the Amiga™

September 22nd, ASDG, Inc. will begin shipping the Convertible™ product line as well as the Mini-Rack-B™

All Convertibles are **100% ZORRO compatible.** In fact, we've designed our board level products to work perfectly even if placed in a sub-standard ZORRO back plane (call our engineers for detailed information).

These features are common to all three:

- **ZERO Wait States - your FAST RAM will really be FAST**
- **Fully ZORRO (100 pin bus) compatible**
- **Fully Auto-Configuring**
- **Four Layer Printed Circuit Board - for quality and reliability**
- **Custom Hi-Speed DRAM Controller**
- **ONE YEAR Parts and Labor Free - in the unlikely event of board failure.**

The Mini-Rack-B (for Budget) converts most ZORRO compatible card cage cards including the Convertibles INTO SIDE MOUNTED cards. The Mini-Rack-B is a two slot card cage with a self contained 6 amp power supply. The packaging is an attractive metal card cage which does not block either mouse port.

If purchased along with a Convertible board product, the Mini-Rack-B will be available (September 22nd) for $150.00. If purchased alone the Mini-Rack-B is $300.00. The entire unit is 10 inches deep by ten inches high and only 6 inches wide.

The Mini-Rack-B coupled with our Convertible product line allows new Amiga owners the ability to start out with an inexpensive card cage and with expansion boards which will be compatible with any future upward expansion. If you had purchased ordinary side mounted products and later decided to expand to a large card cage, your side mounted products would become unusable. With the Convertibles, simply slide them out of the Mini-Rack-B and into the ZORRO back plane of **YOUR** choice.

The three memory boards are being introduced at more than a **TEN PERCENT DISCOUNT** from their list prices. Send us proof that you are a member of any Amiga User Group, and we'll take an **ADDITIONAL FIVE PERCENT** off the list.

The Convertible Memory Board can be ordered NOW:

	LIST PRICE	I INTRODUCTORY PRICE	USER GROUP PRICE
.5 Mbyte of FAST RAM:	**$450.00**	**$395.00**	**$370.00**
1 Mbyte of FAST RAM:	**$650.00**	**$575.00**	**$550.00**
2 Mbyte of FAST RAM	**$900.00**	**$795.00**	**$750.00**

Ordering Information

New Jersey Residents Please Add 6% Sales Tax
ASDG Will Pay UPS Standard Delivery In the Continental U.S.
Other means of shipment are at customer's expense.
All funds In U.S. Currency and draw upon U.S. Banks
Remember, Deliveries Begin September 22nd.
Dealer and VMR Pricing Aviailable
Demonstrations to Large User Groups Can Be Made

Send Checks or Money Orders To:

ADSG , Inc.
280 River Road, Suite #54A
Piscataway, N.J. 08854

AEGIS IMAGES/ANIMATOR

A Review

by **Erv Bobo**

By now, graphics design programs are a staple of the Amiga, but IMAGES, from Aegis Development, has a few differences worth considering - some apparent as soon as the program is booted.

On the black screen before you, a window shows a Fast Menu; this allows rapid point-and-click selection of every tool in the package, but is probably better used once you've become familiar with the program. For right now, drag your cursor to the command bar and click the right mouse button to see the pulldown menus. These are the same commands you saw in the Fast Menu, but here they are much more detailed, words taking the place of icons.

The Project menu allows you to save, load or print a picture as well as adjusting the color palette. Next to it, the Edit menu allows you to Undo your last brush stroke, clear the screen, create a frame or magnify a portion of the work screen. Magnify allows you to work on a small section of your picture one pixel at a time, while Frame allows you to delineate any portion of a painting and save it for later use as a brush or move it to another part of the current picture. As a Frame, it can also be moved to ANIMATOR and be used as a background or foreground object. As if this weren't enough, the area bounded by the Frame may be rotated through 360 degrees.

Next in the line of menus is Special Effects. Here you can turn on Mirrors, for symmetrical drawing along the vertical, horizontal or diagonal - or all three at once. Within this same menu, you can choose Wash, a technique that causes adjacent colors to bleed together; or Smear which does just what it's name implies.

Here, too, you can cause a range of colors (selected from the palette) to Cycle within your painting and this can be used as an elementary type of animation.

A sub-menu to Special Effects shows Pantograph, which copies an area of the painting to another area; Under, which allows you to paint under existing lines or forms; a Grid superimposed on your painting for greater precision; and Spread, which causes the Fill command to use a range of colors rather than the usual single color.

Add to these the color palette, where each color can be adjusted for hue and intensity by slider bars and which contains a variety of pre-made patterns which can be edited to taste; a menu of shapes allowing you to draw freehand or to create a variety of geometric shapes; and a brush menu which shows a wide variety of brush styles and you have a drawing and painting program as complete and as easy to use as any on the market. You will, however, have to bring your own talent to the program.

Now, suppose you do use IMAGES and create a great painting of, say, a landscape ? Beyond printing it, what can you do with it ? Why, simply load it into AEGIS ANIMATOR, use it as a background and create some animation for it.

ANIMATOR is one of the most original programs yet created for the Amiga and one of the most surprising. When you first see how you can draw a simple shape and move it about the screen, then think about how much code would be needed to duplicate that routine on a real computer, you'll be amazed.

Like IMAGES, ANIMATOR requires an Amiga with 512k of RAM. It also resides on a Workbench disk and so can be copied - in fact, the memory requirements engender some special copying instructions. Following them results in a working copy that will always boot to CLI rather than Workbench, thus saving some valuable workspace in RAM.

It is easy to overdo things and run out of memory. Before this happens, ANIMATOR will display a message saying "Memory Panic - System going down!" Before the system does go down, you'll have a chance to save your work.

Because it works from a battery of pull-down menus, with all routines built in, you can get up-and-running on ANIMATOR with only a minimum of instructions. Beginning with the Project menu, you can select a new routine or elect to work with an old one. Then go to the Create menu, which allows you to select from a variety of predefined shapes or to choose a tool for free-hand drawing. The menu of shapes and tools is not as extensive as that in IMAGES, so a bit more work will be required if you are working with complex shapes. An alternative is to create your complex shape in IMAGES, save it as a Window (remember the Frame) and move it to ANIMATOR. As a Window, it can be moved about the screen and it can be embellished with routines created in ANIMATOR.

To begin moving an image, go to the Time menu and select "tweener". A "tween" is a segment of animation and you click on it between segments as a means of keeping everything from happening at once.

Let's say that you've drawn a star and the next step is to move it sideways to the right. Without using tween, the star would simply form itself to the right and there would be no animation. Although at times there may be reasons for simultaneous action, such as having the star appear while rotating on its x axis, a little forethought is necessary in order to make your film turn out the way you hoped it would.

Three-dimensional rotation, of course, is the highest standard of computer animation. Not only is it possible with ANIMATOR, it is also easy. From the Move menu, you may select to Rotate In-plane, around x-axis or around y-axis or, if you don't use tween, around all three simultaneously.

Exploring other options on the menus, we find we can clone a shape, change colors, re-size objects, create a path for a

moving object to follow, outline or fill a shape, change a shape, destroy an object and more. And because we are working with animated rather than static objects, these are all continuous actions.

As you select each option from the menu, a one-line message bar will change to show what that option will do for you. In addition, as you become more adept and more accustomed to the routines indicated by different cursor shapes, you may go to the "quick-select" option, which presents you with an on-screen palette representing every command of which ANIMATOR is capable, much as the Fast Menu does in IMAGES.

When your routine is done, return to the Time menu and select Replay All. Now your creation will run in the proper sequence and, if you really like what you've done, you can select Replay Loop and allow your routine to continually repeat itself while you go drag all your friends and neighbors into the house to show them - or while customers in your showroom are caught by the continous-running display.

Whatever routines you create can be saved to disk and a storyboard option allows you to set up an order in which related routines will be recalled from disk in order to make longer routines possible. We also see this as a terrific tool to be used with a VCR and the composite output of the Amiga, as a means not only of building longer routines on video tape but also of being able to replay them in schools or offices that may not yet have an Amiga.

With the Frame Grabber interface, you'll be able to grab and freeze a frame from videotape, videodisk or direct from your video camera. Thus loaded into ANIMATOR as a background, you can embellish it with whatever animation you can imagine.

Besides video sources, you can load in paintings created with any such program that saves material in the IFF format, a growing standard developed by Electronic Arts - including those created with Deluxe Paint from EA.

In summing up, we would have to say that Aegis ANIMATOR is all we hoped it would be, and more. It is easy to use and it can be deeply explored after only a swift scanning of a few pages. This should not be taken as an invitation to throw away the manual and bull your way through, however. Read it all, refer to it often and you can be turning out animations as good as those seen on the nightly network news.

The price, in a package that includes IMAGES and ANIMATOR along with extensive and easy-to-understand instructions on each, is $139.95 - in our book, a bargain price for one of the best and most complete creative tools available for the Amiga.

Ervin Bobo
23 St. Lawrence
St. Peters, Mo. 63376

•AC•

DELUXE VIDEO CONSTRUCTION SET

A Review

BY JOE LOWERY

Electronic Arts has been known for some time for its meaty programs. DeluxePaint comes to mind as a full course meal and I'm happy to report that Deluxe Video is a feast! This is one of the most mature Amiga products written to date and an immediate classic. However, like all great works of art, definite choices were made and some people will be disappointed with these selections. But let those disgruntled few spend a little time at least appreciating the completeness of Deluxe Video. It is truly AMAZING.

Deluxe Video (programmed by Mike Posehn and Tom Casey of Granite Bay Software) is layed out beautifully. The basis of the program is the hierarchal structure of a video containing certain elements, mainly scenes, and each of those being made up of effects. There is a different window for the overall video and for each scene. A precise timeline allows you to easily overlap specific effects and match their duration. You simply pick up either a "Track" icon or an "Effect" icon and lay it where you wish. Then adjust the starting time, and in many cases, the duration. This is word processing come to video, and fine-tuning your creation is a joy.

One can really get a sense of well thought-out long-range planning here. There is even an option to expand your Part Pool (where many of your video components reside in RAM) if you have more than 512K.

The initial disappointment that I alluded to earlier comes from the cry, "Only 8 colors!" (That's 8 per background and foreground, and... well read on.) Here is where the main choice was made to emphasize smooth, continuous video animation, over a short duration 32 palette one, like Aegis' Animator. Not that one is better for this reason, they are just at different purposes. Deluxe Video is set-up using the Amiga's dual-playfield mode which allows for a foreground to overlap a background (Hacker, Skyfox, and Arcticfox also employ this technique). This mode uses a screen buffer for each playfield so that, effectively, the normal 32 color lo-res palette is split over 4 screens a visible and a hidden each for the background and foreground.

The hidden background allows you to do a dizzying array of wipes as you reveal what lies "behind" the visible background. Figure out the combinations possible with the ability to come from six different directions, each with 5 different stylistic modifiers (such as Slide or Shrink) available, with up to 9 divisions, either horizontally or vertically. I spent an entire evening trying all the variations and I didn't even get close.

The hidden foreground is used as a frame buffer to process the animation sequences in a "page-flipping" technique. This can be turned off with a Strobe effect so that trails of your object are left.

Using Deluxe Video is like programming. In fact, I would say it is quite analagous to audio-visual programming with a compiled language (more about that last phrase in a bit). For *Amazing Computing*, I put together a video that had a firecracker going off in time with a Cajun ditty from Instant Music. I liked that one so much, I added two more, all burning down to the fuse and popping at different rates! There are several "debugging" utilities included in Deluxe Video; a memory map to identify where too much is happening; a clear command; and a stamp feature that moves stationary foreground objects to the background, thus freeing memory overhead.

Now about the "compiled" comment. There is a tremendous amount of disk access involved with Deluxe Video. Going back and forth between viewing your video and working on it is very disk intensive. Deluxe Video does not keep your video and all its constituent parts in memory. Each time you go from the lay-out Video and Scene screens to the viewing screen, Deluxe Video saves your changes to disk in a working file (which means when you are working on your video it takes up twice as much disk space) and then loads the player portion of Deluxe Video and reconstructs your video from disk, loading in each picture as it is requested, fetching each sound, and so on. This insures that the critical timing of loads will be duplicated exactly each time. Then upon returning to your work area, Deluxe Video reloads your most recent lay-out screens. If you're coming from a Basic background to C like me, the parallels to the time spent re-compiling your source code are all to obvious. However, as in C, the results are worth it.

Another specific choice of the programmers was to make this very much a video-based product. Creations can be greatly improved by running them through a VCR at a slower speed and then playing them back normally. An option is available for doing this at either 1/2 or 1/4 speed, thus doubling or quadrupling your frame rate. My experience with this shows that while there is a major benefit, you'll have to re-tune your effects, especially the music and sound, in order to get the best possible results. You can even, if you have access to a single-frame recorder, record at a single-step rate giving you the smoothest possible animations.

In the play mode there is even a VCR remote controller icon enabling you to fast forward, reverse, skip to the end or beginning, mute the sound, or loop continuously. There is also a timer to better help your fine-tuning.

One of the major differences between Deluxe Video and Animator is that here you can incorporate sound and music and that is a big plus. Songs must be in a "SMUS' (Simple Music) IFF files such as from Instant Music & the soon to be released Deluxe Music. Music and sound are given priority over the visual effects so that they never sound draggy. Instruments must be in the proper data drawer when you try to load the songs; there is a menu option which allows you to change those defaults another sign of mature programming. If the instruments cannot be loaded Deluxe Video defaults to its RAM instrument, which, like the manual says, does indeed sound "something like a piano." You can vary the volume and speed of songs, as well as putting a Half of Double Volume Effect to add a dynamic range to them.

Sounds are "8SVX" (8 bit sampled sounds) IFF files digitized sound effects. If the duration chosen is longer than the sound effect, your "plop" or "boing" will repeat until it reaches your selected cut-off point. You can vary the volume, sampled rate, and direction (a sliding bar going from the left to the right speaker). These guys thought of everything!

Deluxe Video also has an Animator-like feature for implementing polygon shapes and text. It is not as full featured as Animator, but still allows you to rotate, size, and move a short text string or any of the 23 different polygon shapes. These range from a square to arrows to a floppy disk shape. With both the text and shapes you have the option of italic, shadow, or outline variations with 8 different pattern fills for the second shadow color.

Another big feature is the level of interactiveness built in to Deluxe Video. There is a KeyWait effect that will delay the video for a specified time or until a specified key is pressed. You can also Chain different videos together giving you videos as long as you have disk storage. Combining those two features is the KeyChain Effect (nice joke, guys). This allows you to branch, upon a specific key being pressed, to any one of 10 videos. And of course each of those could contain a KeyChain, ad interactium...). Here is a mini-Adventure Construction Set tossed in! I can even see how it would allow programmers to map out their games visually before coding!

Finally (whew!) there are a also series of templates for some Quick & Dirty (well, not so dirty really) animation routines that are featured demos to access pie and bar charts data. These will automatically figure the percentages and sizes of the pie slices or bars. They are also nice as "tutorials" or examples of how best to set up your videos.

Let me also add that the manual is very complete and helpful. It would have been nice if they could have alphabetized the reference section, especially since they make such a big deal out of not reading it straight through.

Deluxe Video comes with three disks; besides the Key-Disk Maker you also get an unprotected Player and a Parts & Utilities Disk. The Player has 7 well-done demos on it. Parts & Utilities contains three utilities: Framer for sequential animation using IFF files (DPaint or Images); Unpack for disassembling your videos so that you can re-use their constituent parts; and VidCheck a video compactor that also prints out a breakdown of your video creation including memory usuage, track and effects, parts, etc.

Deluxe Video has the mark of a "meta-program" a program that generates other programs. I predict that not only will we soon be awash in a sea of video, but also that a series of utilities will spring from the fullness of this program. Anybody want to try implementing sprites? How about a video with commercial breaks? "And now back to our show..."

•AC•

Roomers

By The Amigo

Pssst!

Summer is traditionally a slow time in the computer industry; this summer is no exception, even for the Amiga. There hasn't been a lot of activity this past month, mainly because the summer hits are out (Marble Madness, Digi-View, MIDI Gold, etc) and all the developers are looking forward to Comdex/Fall in Las Vegas. This is the BIG show, the one where everybody has CHRISTMAS on his mind. But then again, some things ARE happening.

You may have heard reports that Commodore had decided to mass-merchandise the Amiga. Well, there's a long story behind that one, but the end result is that no, only specialty dealers will be carrying the Amiga.

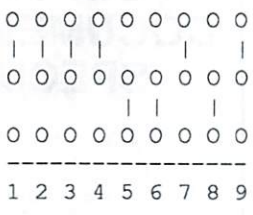
Pascal and Lisp, you say? Metacomco delivered the new versions of both to Commodore quite some time ago. Apparently it is still lost in Quality Assurance. Not actually LOST, mind you -- it's just that they don't feel that it has been shaken out enough for public consumption.

Absoft, makers of AC/Fortran for the Amiga, now have a Basic compiler called (would you believe) AC/Basic. Reportedly, AC/Basic maintains complete compatibility with AmigaBasic, so you won't have to change all your files around to get them compiled.

News from CardCo: First of all, they went out of business. All their C-64 inventory was liquidated by the bank. The aMEGA 1MB board was purchased by a company called C. Ltd, which just happens to be a group of people that used to work for CardCo. The aMEGA RAM board is still shipping. Also coming from CardCo/C. Ltd is a six-slot Zorro expansion box for $499.95 and two things to put into that box: A 2MB memory board for $799.95 and a 20MB DMA hard drive for $799.95. Let's hope we see them!

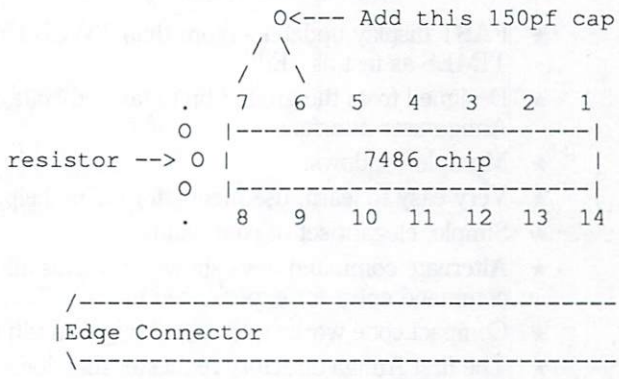
Meanwhile, if you have the aMEGA 1MB board, there are some 'enhancements' you might want to make. These enhancements do not void your warranty (honest!), and if you'd rather have C.Ltd. perform the modifications, you can send it to them and they will do them. The enhancements make your RAM run with no wait states. For the hardware hackers, here are the two fixes.

First, take the cover off the drive and look at the unit. There is a row of straps at the top center of the board next to a large capacitor. It should be strapped like so:



If yours does not look like this -- hooray, you've got a new board. If it does look like this, take an exacto knife and cut strap 5. Then solder a wire so that strap 5 is tied high. Your straps will now look like this:

Note that only strap 5 has changed - nothing else. Now try your board, it should auto-config and pass all its tests. If it does, GREAT! You now have a fast RAM board. If the tests fail, you have to find a 150pf capacitor (or something close) and connect one end to pin 6 of the 7486, the other to pin 7 of the same chip. Here's what it should look like:



Now try your board. If it doesn't work, better call C.Ltd!

In other hardware news, look for Polaroid to link up with a small software firm in producing some Amiga products. I have heard that the Polaroid Palette looks great on the Amiga.

A company called IO Inc has decided to compete with CSA in the 68020 add-on market. They have a 68020 and 68881 board that plugs into the Zorro bus. A software company called Synergy Microsystems is rumored to be preparing a UNIX port to make use of this board. Watch this space for further details.

Metacomco has linked up with a company called Nine Tiles to produce a cheap network interface. It will allow linked Amigas to share file systems. Meanwhile, Ameristar has said that by the time you read this, you can buy an Ethernet interface, along with Sun Microsystem's Network File System (NFS). NFS allows machines to talk to each other and share CPU and file space as if they were one machine. NFS runs under

Roomers

Berkeley UNIX on lots of superminis and supermicros, as well as VAX/VMS and PC-DOS. Of course, Ethernet and its utilities (telnet, ftp) are much more standard, and appear on most machines that support Ethernet. Also from Ameristar is a token passing network.

Look for a new ALINK soon; one that supports relative addressing. That should bring the size of executable programs down considerably. The Manx Aztec C linker supports relative addressing now; that's why executables compiled and linked under Manx C are smaller than those under Lattice C. A new ALINK should decrease that gap.

For you COMAL fans, Unicomal in Sweden is porting it to the Amiga. Also being ported is Precision Software's Superscript program.

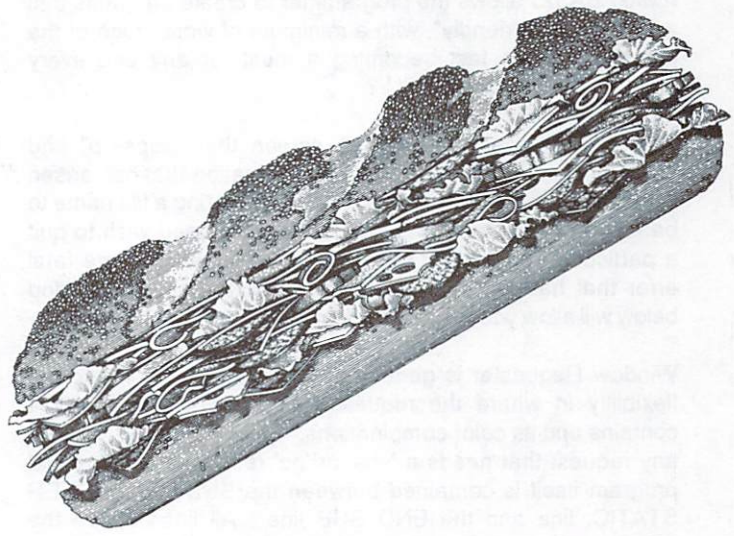
Hopefully things will pick up next month.

•AC•

The AmigaBasic Tutorial

Subroutines

by **Kelly Kauffman**
Compuserve [70206,640]



Subroutines are primarily what a computer is really about. Computers are great for doing monotonous things over and over again.

Well, that's what a subroutine is all about. It allows the computer to do a set series of steps on data--over and over again. Subroutines exist in programs to be referred to many times. A simple example of this would be a routine to add a score to a players points in a game, while updating the hi-score. Instead of having lines all over your program saying something like..

"SCORE=SCORE+POINTS" and
"IF SCORE>HISCORE THEN HISCORE=SCORE."

With that method, you would have LOTS of extra lines in your program which takes away precious memory. Instead, the alternative... subroutines!

Now in place of these lines all over the place, we have one area of the program set away to do just that...keep track of scores. Here would be an example of the subroutine:

```
SCOREKEEP:

SCORE=SCORE+POINTS
   IF SCORE>HISCORE THEN HISCORE=SCORE
RETURN
```

That's a subroutine. They start by being named. This is done by typing a name and then following it with a colon. They end with the command "RETURN." To go to a routine (call a routine), you simply would say:

```
GOSUB SCOREKEEP:
```

MBasic will automatically jump to wherever in the program SCOREKEEP: exists, execute all the statements between SCOREKEEP: and the RETURN statement, and then upon executing the RETURN statement, jumps back immediately to the NEXT command following the original GOSUB statement, and program execution continues normally.

It's also a good idea when using subroutines, to locate them at the bottom of the program. If the subroutines are located within the program, MBasic will NOT jump over them just because they're subroutines, it will however, execute them step by step, and then generate an error when it tries to execute the RETURN statement. This happens because there was no GOSUB statement to "legally" enter the subroutine. Also, the last line of your program, that is, the last line before the subroutines begin, should be "END" to stop MBasic from executing, there by keeping it from stumbling into the subroutines.

When data is changed within a subroutine, such as our SCORE's, and then RETURNed out of the subroutine, the data will NOT go back to it's original values before it entered the subroutine. The values will be the new changed values that the subroutine performed on the data.

Subroutines can be used for much more than just this rudimentary example. They can, for instance, be used in menu driven programs (menu driven meaning programs that present a menu of options, and then let the user select an option by pressing a key,) to execute complete commands and then RETURN to the main menu once the operation has been completed. Subroutines also make it nice to label the different parts of your program for easy identification and de-bugging. They are probably one of the best bug-fighting elements of a program that you can use.

One word of caution when using subroutines though...Make absolutely, positively, sure that that the steps you perform on the data are correct, because if they aren't....big, BIG, troublems can arise!!!! (troublems=troubles+problems.)

When subroutines have been fine-tuned and are working fine, they can greatly reduce the time it takes to develop a good, working piece of software.

•AC•

Window Requesters
in Amiga Basic

By **Steve Michel**

AMIGABASIC allows the programmer to create programs that are very "user-friendly", with a minimum of work. One of the friendlies that is fast becoming a must for any and every program is the 'requester.

A requester is an area of the screen that "pops-up" and prompts the user for a response to a situation that has arisen in the program. This may range from requesting a file name to be loaded, confirming that the user does indeed wish to quit a particular application, or informing the user of some fatal error that has just occurred. The program listing included below will allow you to easily add this touch to your programs.

Window Requester is generic in that it allows for maximum flexibility in where the requestor appears, what prompt it contains and its color combinations. It is designed to work for any request that needs a 'yes' or 'no' response. The actual program itself is contained between the SUB REQUESTER STATIC: line and the END SUB line. All lines above the STOP command are a simple driver to show the ease of use of Window Requester. Window Requester is actually a SUBPROGRAM. This means that although it is contained within the framework of a larger program, when it is called by the larger program, it acts autonomously. All variables contained within the SUBPROGRAM are known only to the SUBPROGRAM, and are thus referred to as 'local' variables. The net effect of this situation is that the SUBPROGRAM can be incorporated "as-is" into several different programs without fear of the SUBPROGRAM corrupting variables with the same name in the larger program. It also means that we can cut down on the amount of programming that we as programmers must do.

Once a generalized SUBPROGRAM has been developed and debugged, it can be stored on disk and then added to any program that requires it. Ideally then, the programmer could begin creating a whole library of specialized SUBPROGRAMS and simply add them to the program that is currently being created. This concept of programming is often referred to as modular programming. Being completely independent sometimes has its drawbacks and thus a way of communicating information between the SUBPROGRAM and the larger program has been provided by the SHARED statement as indicated in the second line of the SUBPROGRAM example. Any data that is to be passed to or from the SUBPROGRAM must be passed through a variable listed in a SHARED statement. Any variable so defined becomes known to both the larger program and the SUBPROGRAM.

In this example, this includes all the information needed to make the requester function as desired. Each variable is described in REM statements at the beginning of the listing and are fairly self-explanatory.

A word about the color numbers, however, may be appropriate. Color numbers zero through three are listed as the default values in the driver, assuming that the larger program is working within a standard AMIGABASIC screen which has a depth of two. If your application has a deeper screen, the color choices may be increased as allowed. Color zero is the default screen background color (usually blue) and should not be used to print a message as it will not be seen against the same color background.

The SUBPROGRAM works by opening a window within the current screen and then directing output to that window. This is accomplished by the WINDOW and WINDOW OUTPUT statements.

Once output is directed to a window, all locating statements (LINE, LOCATE, PAINT) are relative to the upper left hand corner of the window and NOT the screen in which the window is open. The location of the window within the screen is specified by the variables REQX1 and REQY1 and passed from the calling program. After the requester is drawn, the routine WAITER then waits for the mouse to be positioned and clicked over either the "YES" or "NO" response. The SUBPROGRAM will return the selected answer to the calling program via the variable CHOICE$. Just before exiting the SUBPROGRAM, a WINDOW CLOSE command causes the window and all the displayed data to be removed from the screen. The important feature of this windowing technique then becomes apparent. The information on the main screen that was covered by the requester, is now redrawn as though nothing had occurred. All this is done automatically by the operating system.

Once the program is entered and saved to disk as a regular AMIGABASIC program, it should also be saved in ASCII format. This is required if it is to be added as a SUBPROGRAM to a larger program at some later time.

To save the program in this manner, enter on the output window SAVE "WINDOW_REQUESTER",A. Make sure the ,A is outside the quotes. You may also select SAVE AS from the menu, again being careful to use the quotes correctly.

When the SUBPROGRAM module is needed, it may be added to an existing program by using the MERGE command. The correct syntax is MERGE "filename", where filename is the name of a file saved in ASCII format. The SUBPROGRAM will be appended to the end of the program currently in memory. It may then be moved with the CUT and PASTE operations. How easy can it get?

I hope this small piece of coding will enhance your program appearance and speed along your program development.

Program Listing:

```
REM  Window Requester by Steve Michel

REM  call 'REQUESTER' whenever a
REM  YES or NO answer is required
REM' set following variables to desired values
before calling
' reqx1   = x position of top left-hand corner of
requester box
' reqy1   = y position of top left-hand corner of
requester box
' backcol = background color of requester box
' msgcol  = color of written message in box
' outcol  = requester outline color
' msg$    = message prompt to be displayed
'                -- 22 characters maximum
' titles$ = the title to appear in the requester
window after returning from requester, the variable
choice$ will equal "YES" or "NO" reflecting the
user's selection

  CLS
  INPUT "starting x location"; reqx1
  INPUT "starting y location"; reqy1
  INPUT "background color (0-3)"; backcol
  INPUT "message color (0-3)"; msgcol
  INPUT "outline color (0-3)"; outcol
  INPUT "title";title$
  INPUT "message string"; msg$
  CALL REQUESTER
  LOCATE 20,1: PRINT "THE USER CHOSE ... ";
CHOICE$
  STOP

SUB REQUESTER STATIC:
SHARED title$, msg$, reqx1, reqy1, backcol, msgcol,
outcol, CHOICE$
  reqx2 = reqx1 + 206: reqy2 = reqy1 + 47
  yesx = 23: yesy = 26: nox = 134: noy = yesy
  WINDOW 2,title$,(reqx1,reqy1)-(reqx2,reqy2),0
  WINDOW OUTPUT 2: PAINT (100,20),backcol
  msgpad$ = " " + LEFT$(msg$,22) + " "
  msglen = LEN(msgpad$)
  xloc = INT((24-msglen)/2) + 1:xline = (xloc-1)*8
  COLOR msgcol: LOCATE 2,xloc: PRINT  msgpad$;
  LINE (xline,7)-(xline+8*msglen-1,7),0
  LINE (yesx,yesy)-(yesx+57,yesy+18),outcol,bf
  COLOR msgcol: LOCATE 5,5: PRINT " YES ";
  LINE (32,31)-(71,31),0
  LINE (nox,noy)-(nox+50,noy+18),outcol,bf
  LINE (144,31)-(175,31),0
  LOCATE 5,19: PRINT " NO ";

WAITER:
  CHOICE$ = "none"
  WHILE MOUSE(0) <> 1
  WEND
  xpos = MOUSE(3): ypos = MOUSE(4)
  IF ypos < yesy OR ypos > yesy+18 THEN WAITER
  IF xpos >= yesx AND xpos <= yesx+54
     THEN CHOICE$ = "YES"
  IF xpos >= nox AND xpos <= nox+48 THEN CHOICE$ =
"NO"
  IF CHOICE$ = "none" THEN WAITER
  WINDOW CLOSE 2
END SUB
```

•AC•

ECE MIDI
MUSICAL INSTRUMENT DIGITAL INTERFACE

The ECE MIDI plugs into the serial port on the AMIGA and allows it to communicate with midi compatible musical instruments and equipment. For your convenience the ECE MIDI has a RS-232 bypass port so you do not have to disconnect your equipment when you want to use a modem or a serial printer. Encased in a small attractive Amiga bone colored box, the ECE MIDI has In, Out, and Thru ports.

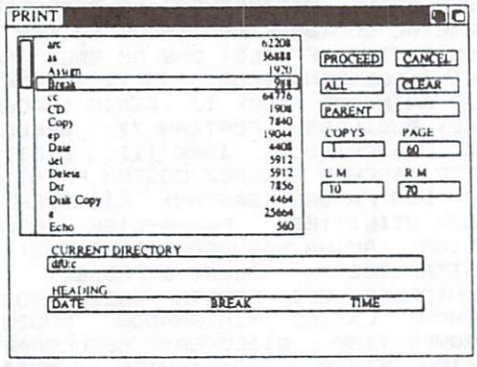
Suggested retail: $59.95

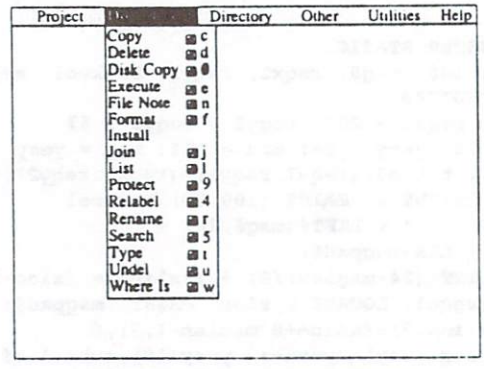
MIDI
Research & Development

SPEEDY AMIGOS
A COLLECTION OF DOS UTILITIES FOR THE AMIGA

SPEEDY AMIGOS, developed with both the novice and experienced user in mind, contains over 50 handy DOS routines that can be quickly and easily executed with the mouse or keyboard. Pull-down menus, windowing, and built in requestors assure that syntax errors are kept to a minimum. Several existing AmigosDOS routines have been enhanced and new features, such as Spooler, Where Is, Undelete, and Function Keys, have been added. Fast and efficient, SPEEDY AMIGOS simplifies disk operation utilities and enhances the Amiga user interface.

Suggested retail: $64.95





AMIGA REF
A QUICK REFERENCE CARD FOR AMIGOSDOS VERSION 1.1

In AMIGA REF, DOS commands are logically grouped by type - DISK, FILE, UTILITY, and BATCH - to help you quickly find the command or syntax that you need. Tips on getting started, examples of usage for AmigaDOS routines, and a complete listing of WILDCARDS, DEVICE and CONTROL CODES are included. Advice on working with a RAM DISK and getting AMIGA HELP are also included.

Suggested retail: $2.95

Research & Development

ECE Research & Development Corp.
1651 N. Monroe Street
Tallahassee, Fl. 32303

904-681-0786

Dealer Inquires Welcome

ROT

By Colin French

If you're itching to experiment with 3D graphics on your Amiga, try this AmigaBasic program. With ROT you can create an object composed of up to 95 points and 95 filled polygons. All you need is at least 512K, a mouse, and the patience to find the mistakes you make while typing it in!

(ROT is available on an Amicus disk. The disk also contains several demo objects.) ROT requires a lot of memory. If your WorkBench does not show at least 380,000 bytes free, do not try to run it. A normal two drive 512K system should have this much available after booting up the WorkBench disk. A file that came on the AmigaBasic disk called 'graphics.bmap' must be on the same disk as ROT.

AN OVERVIEW

In ROT an object is composed of filled polygons, like the faces of a cube. A polygon is created by selecting the points used as its vertices. Each polygon must have at least three vertices but no more than six. A different color can be assigned to each polygon. Careful choice of hues can produce a shaded effect. Once an object is created, a series of frames are drawn each with the object in a different position. The frames are captured as bitmapped images. By showing them quickly in sequence the object will go through an animated action. In every frame, you specify the rotations and translations of the object along all three axes. For example, using a larger and larger Y axis rotation in each frame produces a spinning action when the sequence is played back.

EDITING OBJECTS

When ROT is first started the Object Editor screen is displayed. You can switch to the Action Editor by selecting the first item in the ACTION menu. To get back, just choose the top item in the OBJECT menu. On the left of the Object Editing screen are three views of your object showing its top, side, and front. To see how these fit together, imagine folding the top and front views away from you until their edges touch. You now have a half-cube that surrounds your object. Any rotations applied to the object will take place around the center of this cube. In all three views a green circle highlights the currently selected point. Two points will often appear to be right on top of one another. Carefully check the position of the highlight in all three views to make sure you've selected the point you really want.

A point with all three coordinates set to zero is considered nonexistent and is not displayed. Clicking in a view will change two of the point's coordinates so it ends up under the cursor. Which two coordinates are changed depends on the view in which you click. By doing this in at least two views you can position the point exactly where you want it in 3D space. The edges of the currently selected polygon are highlighted in orange. Of course, if it does not have any vertices chosen for it yet then the polygon doesn't exist and no highlighting appears.

On the upper left of the Object Editing screen is a block of controls that deal with points. At the top is a slider for picking the point to modify. Click on the arrows to go to the next or previous point, or anywhere within the slider to quickly skip through the list. The point you select is highlighted in the three views on the left and its coordinates can be changed by clicking within the views. To quickly set all the coordinates of a point to zero, click on the 'ZERO POINT' box. This becomes a nonexistent point, so be careful that no polygon was using it as a vertex.

Below the point controls is another set used to edit polygons. A polygon is created by selecting the points that form its vertices. Use the points slider to highlight the point you want then click on 'ADD POINT' to use it as a vertex. A polygon can have between three and six vertices. Its edges are highlighted in orange in the three views. The order in which you specify the points is important. Go in one direction around the perimeter of the polygon. If you see the edges crossing you'll know the points are out of order. Click on 'UNDO POINT' to back up through the list of vertices until the problem is eliminated. To get rid of the polygon entirely click on 'DELETE POLYGON'.

Down at the bottom of the screen is a color palette. The color that will be used to draw the currently selected polygon is highlighted with an orange rectangle. To change it, click on the color you want. By selecting items in the OBJECT menu you can save the database of your 3D object, load a previously saved object, or erase the object and start over. The files saved will have the suffix '.ROTOBJ' appended to the name you supply. When loading an object do not type this suffix, just the filename. If you forget the name of a file, select 'Files' from the ROT menu and try to spot it as the list scrolls by!

EDITING ACTIONS

Pick the first item in the ACTION menu to display the Action Editing screen. The top section is where your object will be drawn and along the bottom are several controls. Use the slider on the left to select the frame to work with. In each frame you can set the rotations and translations of the object along all three axes. To change a particular value, click on it and type the number you want. Values are checked to make sure they fall within acceptable limits. An orange arrow will appear to let you know that the frame needs to be redrawn.

Clicking on the 'REDRAW FRAME' button will recalculate the image in the frame so it corresponds to the factors you've set. Then advance to the next frame, set its factors and redraw its image. Continue in this manner until all 12 frames are done. If you go back to the Object Editor screen, modify the object and then return, the frames will not match the revised object. To update them you could advance to each one and click on 'REDRAW FRAME' or you can use 'REDRAW ALL' to do each in sequence. Once all 12 frames have been drawn, click on 'PLAY' to show the animation sequence. Adjust the

speed slider as required. To show the frames continuously, look under the ACTION menu and select 'REPEAT AT END'. A checkmark next to this item indicates when it is activated.

Another option in the menu is 'REVERSE AT END'. If you choose this the frames will be shown from first to last and then back to the first again. Both these options can be combined. To halt a continuously running sequence, click on 'STOP'. Other items in the ACTION menu will save the object's action, load an action from disk, or erase the current action and start over. The file that is saved does not include the bitmapped frames, just the factors that are used to draw them. When you load an action, click on 'REDRAW ALL' to regenerate the frames. They are not saved because they occupy almost 109K. It would require over six minutes to load or save them under AmigaBasic. Oh please oh mighty MicroSoft, give us BLOAD and BSAVE commands next time!

The final item in the ACTION menu is 'CALC BETWEEN...' . This will calculate and draw the frames that lie between any two that you specify. For example, set the Y rotation of frame one to 90 degrees and that of frame seven to 180. Since you want to take six steps to go from 90 degrees to 180, each frame inbetween will have a Y rotation 15 degrees greater than the previous one. You could do this yourself by advancing one frame at a time, setting the Y rotation and redrawing it, but 'CALC BETWEEN...' automates the procedure. The direction of rotation is always chosen to move the object through the smallest angle possible. If you set the starting frame to zero degrees and the ending frame to 270, then the object will be rotated -90 degrees, not + 270 degrees.

HINTS AND CAVEATS

The first time any AmigaBasic routine is used, it is very slow. Subsequent calls to the same part of the program execute at a normal speed. There are some parts of ROT, such as the input routine for filenames, which are so slow at first that you might suspect the program has 'hung'. Exercise a bit of patience before pummelling the computer. When the frames are played back there is a fair amount of flicker. This may be caused by basic drawing the bitmapped images in the middle of a screen scan. Sometimes changing the colors of an object helps minimize this problem.

The X and Y translations are not true 3D transformations. Instead they indicate where to draw the frame on the screen. This was done to reduce the size of the images and therefore the amount of memory required to store them. As each succesive frame is displayed it overlays the previous one and thereby erases it. If you use too great an X axis translation with a large object then parts of it may not be erased properly. Either make a smaller object (use a Z translation to reduce its apparent size) or smaller changes in X translation.

The ROT program itself fills the standard 25K basic workspace. I had to delete most of the comments to get it to fit. Something else that got squeezed out is error checking during disk I/O, so be sure you know the correct name of a file when you load it. Usethe 'Files' option in the ROT menu to check what's on the disk.

```
GOTO main:
intro:
PRINT"                    ROT                "
PRINT"      -------------------------- "
PRINT"      By: Colin French     May 1986 "
PRINT"      Last Revision:  19/06/86 CJF "
PRINT"      -------------------------- "
PRINT"         Copying is encouraged!     "
PRINT
PRINT
PRINT"  Note: AmigaBasic routines are very "
PRINT"  slow the first time they are used.  "
PRINT"  Be patient if ROT seems to 'hang'."
RETURN

main:
  IF FRE(0)<100000& THEN CLEAR,150000&
  GOSUB init
  quit=0
  WHILE NOT(quit)
    b=MOUSE(0)
    x=MOUSE(1)
    y=MOUSE(2)
    IF b<>0 THEN
      IF objscr THEN GOSUB edobj
      IF actscr THEN GOSUB edact
    END IF
    m=MENU(0)
    i=MENU(1)
    IF m<>0 THEN GOSUB menuchk
    z$=INKEY$
    IF z$<>"" THEN GOSUB keychk
  WEND
  GOSUB cleanup
END

menuchk:
  ON m GOSUB rotmenu,objmenu,actmenu
RETURN
rotmenu:
  IF i=1 THEN GOSUB listfiles
  IF i=3 THEN quit=(-1)
RETURN
objmenu:
  IF i=1 AND objscr=0 THEN
    objscr=(-1)
    actscr=0
    MENU 2,1,2
    MENU 2,2,1
    MENU 2,3,1
    MENU 2,4,1
    MENU 3,1,1
    MENU 3,2,0
    MENU 3,3,0
    MENU 3,4,0
    MENU 3,6,0
    MENU 3,7,0
    MENU 3,8,0
    GOSUB drw.objscr
  END IF
  IF i=2 THEN GOSUB loadobj
  IF i=3 THEN GOSUB saveobj
  IF i=4 THEN GOSUB newobj
RETURN
actmenu:
  IF i=1 AND actscr=0 THEN
    actscr=(-1)
    objscr=0
    MENU 3,1,2
    MENU 3,2,1
    MENU 3,3,1
    MENU 3,4,1
    MENU 3,6,1+ABS(actrpt)
    MENU 3,7,1+ABS(actrev)
    MENU 3,8,1
    MENU 2,1,1
    MENU 2,2,0
    MENU 2,3,0
    MENU 2,4,0
    FOR n=1 TO 12
      frmchg(n)=1
    NEXT
    GOSUB drw.actscr
  END IF
  IF i=2 THEN GOSUB loadact
  IF i=3 THEN GOSUB saveact
  IF i=4 THEN GOSUB newact
  IF i=6 THEN
    actrpt=NOT(actrpt)
    MENU 3,6,1+ABS(actrpt)
  END IF
  IF i=7 THEN
    actrev=NOT(actrev)
    MENU 3,7,1+ABS(actrev)
  END IF
  IF i=8 THEN GOSUB calctween
RETURN

listfiles:
  s$="Files on:"
  GOSUB drw.filereq
  s$="DF0:"
  GOSUB getstring2
  IF s$<>"c" AND s$<>"C" AND s$<>"" THEN
    CLS
    FILES s$
    PRINT
    GOSUB click.continue
    IF objscr<>0 THEN
      GOSUB drw.objscr
    ELSE
      GOSUB drw.actscr
    END IF
    GOSUB nobut
  END IF
RETURN

loadobj:
  s$="Load:"
  GOSUB drw.filereq
  GOSUB getstring
  IF s$<>"C" AND s$<>"c" AND s$<>"" THEN
    s$=s$+".ROTOBJ"
    OPEN s$ FOR INPUT AS #1
    FOR n=0 TO 95
      FOR n2=0 TO 3
        INPUT#1,pt(n,n2)
      NEXT
    NEXT
    FOR n=0 TO 95
      FOR n2=0 TO 6
        INPUT#1,poly(n,n2)
      NEXT
    NEXT
    FOR n=0 TO 95
      INPUT#1,polyclr(n)
    NEXT
    FOR n=0 TO 95
      INPUT#1,vrt(n)
    NEXT
```

```
    CLOSE #1
    pt=1
    poly=1
  END IF
  GOSUB drw.objscr
RETURN

saveobj:
  s$="Save:"
  GOSUB drw.filereq
  GOSUB getstring
  IF s$<>"C" AND s$<>"c" AND s$<>"" THEN
    s$=s$+".ROTOBJ"
    OPEN s$ FOR OUTPUT AS #1
    FOR n=0 TO 95
      FOR n2=0 TO 3
        PRINT#1,pt(n,n2);
      NEXT
    NEXT
    FOR n=0 TO maxpoly
      FOR n2=0 TO 6
        PRINT#1,poly(n,n2);
      NEXT
    NEXT
    FOR n=0 TO maxpoly
      PRINT#1,polyclr(n);
    NEXT
    FOR n=0 TO maxpoly
      PRINT#1,vrt(n);
    NEXT
    CLOSE #1
  END IF
  GOSUB drw.objscr
RETURN

newobj:
  s$=" erase current object?"
  GOSUB you.sure
  IF sure THEN
    FOR n=0 TO maxpt
      FOR n2=0 TO 2
        pt(n,n2)=0
      NEXT
    NEXT
    FOR n=0 TO maxpoly
      FOR n2=0 TO 6
        poly(n,n2)=0
      NEXT
      polyclr(n)=0
      vrt(n)=0
    NEXT
    pt=1
    poly=1
  END IF
  GOSUB drw.objscr
RETURN

loadact:
  s$="Load:"
  GOSUB drw.filereq
  GOSUB getstring
  IF s$<>"C" AND s$<>"c" AND s$<>"" THEN
    s$=s$+".ROTACT"
    OPEN s$ FOR INPUT AS #1
    FOR n=1 TO 12
      INPUT#1,xrot(n),yrot(n),zrot(n)
      INPUT#1,xtran(n),ytran(n),ztran(n)
    NEXT
    INPUT#1,spd,actrpt,actrev
    CLOSE #1
    frm=1
    FOR n=1 TO 12
      frmchg(n)=1
    NEXT
    GOSUB drw.frmnum
    GOSUB drw.spdnum
    GOSUB drw.factors
    GOSUB drw.update
    MENU 3,6,1+ABS(actrpt)
    MENU 3,7,1+ABS(actrev)
  END IF
  LINE(0,0)-(311,131),0,bf
  GOSUB putfrm
RETURN

saveact:
  s$="Save:"
  GOSUB drw.filereq
  GOSUB getstring
  IF s$<>"C" AND s$<>"c" AND s$<>"" THEN
    s$=s$+".ROTACT"
    OPEN s$ FOR OUTPUT AS #1
    FOR n=1 TO 12
      PRINT#1,xrot(n);yrot(n);zrot(n);
      PRINT#1,xtran(n);ytran(n);ztran(n);
    NEXT
    PRINT#1,spd;actrpt;actrev;
    CLOSE #1
  END IF
  LINE(0,0)-(311,131),0,bf
  GOSUB putfrm
RETURN

newact:
  s$=" erase current action?"
  GOSUB you.sure
  IF sure THEN
    FOR n=0 TO 12
      xrot(n)=0:yrot(n)=0:zrot(n)=0
      xtran(n)=0:ytran(n)=0:ztran(n)=0
    NEXT
    spd=20
    actrpt=0
    actrev=0
    MENU 3,6,1
    MENU 3,7,1
    frm=1
  END IF
  GOSUB drw.actscr
RETURN

calctween:
  GOSUB gettween
  IF stfrm>endfrm THEN SWAP stfrm,endfrm
  stp=endfrm-stfrm
  LINE(0,0)-(311,131),0,bf
  IF stp>1 THEN
    xrot=xrot(endfrm)-xrot(stfrm)
    IF xrot>180 THEN xrot=(360-xrot)*(-1)
    IF xrot<-180 THEN xrot=xrot+360
    stpxrot=xrot/stp
    yrot=yrot(endfrm)-yrot(stfrm)
    IF yrot>180 THEN yrot=(360-yrot)*(-1)
    IF yrot<-180 THEN yrot=yrot+360
    stpyrot=yrot/stp
    zrot=zrot(endfrm)-zrot(stfrm)
    IF zrot>180 THEN zrot=(360-zrot)*(-1)
    IF zrot<-180 THEN zrot=zrot+360
    stpzrot=zrot/stp
    xtran=xtran(endfrm)-xtran(stfrm)
```

```
    stpxtran=xtran/stp
    ytran=ytran(endfrm)-ytran(stfrm)
    stpytran=ytran/stp
    ztran=ztran(endfrm)-ztran(stfrm)
    stpztran=ztran/stp
    FOR frm=stfrm+1 TO endfrm-1
      n=frm-stfrm
      xrot(frm)=xrot(stfrm)+INT(stpxrot*n)
      IF xrot(frm)>359 THEN xrot(frm)=xrot(frm)-
360
      IF xrot(frm)<0 THEN xrot(frm)=xrot(frm)+360
      yrot(frm)=yrot(stfrm)+INT(stpyrot*n)
      IF yrot(frm)>359 THEN yrot(frm)=yrot(frm)-
360
      IF yrot(frm)<0 THEN yrot(frm)=yrot(frm)+360
      zrot(frm)=zrot(stfrm)+INT(stpzrot*n)
      IF zrot(frm)>359 THEN zrot(frm)=zrot(frm)-
360
      IF zrot(frm)<0 THEN zrot(frm)=zrot(frm)+360
      xtran(frm)=xtran(stfrm)+INT(stpxtran*n)
      ytran(frm)=ytran(stfrm)+INT(stpytran*n)
      ztran(frm)=ztran(stfrm)+INT(stpztran*n)
      GOSUB drw.frmnum
      GOSUB drw.factors
      GOSUB high.redraw
      GOSUB drw.frame
      GOSUB getfrm
      frmchg(frm)=0
      GOSUB drw.update
      GOSUB unhigh.redraw
    NEXT
    frm=stfrm
    GOSUB drw.frmnum
    GOSUB drw.factors
  END IF
  LINE(0,0)-(311,131),0,bf
  GOSUB putfrm
RETURN

gettween:
  GOSUB drw.tweenreq
  maxchar=2
  xt=188:yt=74
  GOSUB getstring
  stfrm=VAL(s$)
  IF stfrm<1 THEN stfrm=1
  IF stfrm>12 THEN stfrm=12
  yt=82
  GOSUB getstring
  endfrm=VAL(s$)
  IF endfrm<1 THEN endfrm=1
  IF endfrm>12 THEN endfrm=12
RETURN

drw.tweenreq:
  LINE(58,48)-(254,92),0,bf
  LINE(60,50)-(252,90),3,bf
  LINE(61,51)-(251,89),2,bf
  CALL move&(rp&,76,66)
  PRINT"Calculate Inbetween"
  CALL move&(rp&,92,74)
  PRINT"From frame:"
  CALL move&(rp&,108,82)
  PRINT"To frame:"
RETURN

drw.filereq:
  LINE(50,48)-(262,92),0,bf
  LINE(52,50)-(260,90),3,bf
  LINE(53,51)-(259,89),2,bf
    CALL move&(rp&,60,66)
    PRINT"Temporary File Requestor"
    CALL move&(rp&,92,74)
    PRINT"('C' to Cancel)"
    CALL move&(rp&,60,82)
    PRINT s$
    xt=68+LEN(s$)*8
    yt=82
    maxchar=23-LEN(s$)
  RETURN

  you.sure:
    GOSUB drw.surereq
    GOSUB nobut
    answer=0
    WHILE NOT(answer)
      b=MOUSE(0)
      x=MOUSE(1)
      y=MOUSE(2)
      IF b<>0 THEN
        IF y>80 AND y<92 THEN
          IF x>75 AND x<127 THEN
            sure=(-1)
            answer=(-1)
          END IF
          IF x>187 AND x<239 THEN
            sure=0
            answer=(-1)
          END IF
        END IF
        GOSUB nobut
      END IF
    WEND
  RETURN
```

```
drw.surereq:
  LINE(43,48)-(270,100),0,bf
  LINE(45,50)-(268,98),3,bf
  LINE(46,51)-(267,97),2,bf
  CALL move&(rp&,53,66)
  PRINT"Are you SURE you want to"
  CALL move&(rp&,53,74)
  PRINT LEFT$(s$,24)
  LINE(75,80)-(127,92),1,b
  LINE(187,80)-(239,92),1,b
  CALL move&(rp&,93,89)
  PRINT"OK"
  CALL move&(rp&,190,89)
  PRINT"CANCEL"
RETURN

keychk:
  RETURN

edobj:
  IF x>189 AND x<306 THEN
    IF y>24 AND y<32 THEN GOSUB ptslider
    IF y>82 AND y<90 THEN GOSUB polyslider
    IF y>36 AND y<48 THEN GOSUB zeropt
    IF y>94 AND y<106 THEN GOSUB addpt
    IF y>110 AND y<122 THEN GOSUB undopt
    IF y>126 AND y<138 THEN GOSUB delpoly
    IF y>142 AND y<171 THEN GOSUB selclr
  END IF
  IF x>3 AND x<82 AND y>3 AND y<82 THEN
    pt(pt,2)=(x-vx1)*(-1)
    pt(pt,0)=(y-vy1)*(-1)
    GOSUB drw.views
    GOSUB nobut
  END IF
  IF x>3 AND x<82 AND y>96 AND y<175 THEN
    pt(pt,2)=(x-vx2)*(-1)
    pt(pt,1)=(y-vy2)*(-1)
    GOSUB drw.views
    GOSUB nobut
  END IF
  IF x>97 AND x<176 AND y>96 AND y<175 THEN
    pt(pt,0)=x-vx3
    pt(pt,1)=(y-vy3)*(-1)
    GOSUB drw.views
    GOSUB nobut
  END IF
RETURN

zeropt:
  p=pt
  GOSUB unhigh.pt
  GOSUB erase.pt
  pt(pt,0)=0
  pt(pt,1)=0
  pt(pt,2)=0
  GOSUB drw.pt
  GOSUB high.pt
RETURN

addpt:
  IF vrt(poly)>5 THEN BEEP:RETURN
  IF pt(pt,0)=0 AND pt(pt,1)=0 AND pt(pt,2)=0 THEN
RETURN
  vrt(poly)=vrt(poly)+1
  poly(poly,vrt(poly))=pt
  GOSUB drw.views
  GOSUB nobut
RETURN

undopt:
  IF vrt(poly)>0 THEN
    poly(poly,vrt(poly))=0
    vrt(poly)=vrt(poly)-1
  END IF
  GOSUB drw.views
  GOSUB nobut
RETURN

delpoly:
  IF vrt(poly)>0 THEN
    c=polyclr(poly)
    GOSUB unhigh.clr
    FOR n=0 TO vrt(poly)
      poly(poly,n)=0
    NEXT
    vrt(poly)=0
    polyclr(poly)=0
    c=polyclr(poly)
    GOSUB high.clr
  END IF
  GOSUB drw.views
  GOSUB nobut
RETURN

selclr:
  IF x>191 AND x<304 THEN
    c=INT((x-192)/14)+INT((y-143)/7)*8
    IF c<>polyclr(poly) THEN
      GOSUB high.clr
      SWAP c,polyclr(poly)
      GOSUB unhigh.clr
    END IF
  END IF
RETURN
```

```
drw.views:
  GOSUB erase.views
  p=pt
  GOSUB high.pt
  FOR p=1 TO maxpt
    GOSUB drw.pt
  NEXT
  p=pt
  t=poly
  FOR poly=1 TO maxpoly
    GOSUB drw.poly
  NEXT
  poly=t
  GOSUB high.poly
RETURN

erase.views:
  LINE(2,2)-(83,83),0,bf
  LINE(2,95)-(83,176),0,bf
  LINE(96,95)-(177,176),0,bf
RETURN

drw.pt:
  IF pt(p,0)<>0 OR pt(p,1)<>0 OR pt(p,2)<>0 THEN
    PSET(vx1-pt(p,2),vy1-pt(p,0))
    PSET(vx2-pt(p,2),vy2-pt(p,1))
    PSET(vx3+pt(p,0),vy3-pt(p,1))
  END IF
RETURN

erase.pt:
  COLOR 0
  GOSUB drw.pt
  COLOR 1
RETURN

high.pt:
  CIRCLE(vx1-pt(p,2),vy1-pt(p,0)),2,2
  CIRCLE(vx2-pt(p,2),vy2-pt(p,1)),2,2
  CIRCLE(vx3+pt(p,0),vy3-pt(p,1)),2,2
RETURN

unhigh.pt:
  CIRCLE(vx1-pt(p,2),vy1-pt(p,0)),2,0
  CIRCLE(vx2-pt(p,2),vy2-pt(p,1)),2,0
  CIRCLE(vx3+pt(p,0),vy3-pt(p,1)),2,0
RETURN

drw.poly:
  IF vrt(poly)>0 THEN
    PSET(vx1
pt(poly(poly,1),2),vy1pt(poly(poly,1),0))
    PSET(vx2-pt(poly(poly,1),2),vy2-
pt(poly(poly,1),1))
    PSET(vx3+pt(poly(poly,1),0),vy3-
pt(poly(poly,1),1))
    IF vrt(poly)>1 THEN
      FOR n=2 TO vrt(poly)
        LINE(vx1-pt(poly(poly,n-1),2),vy1-
pt(poly(poly,n-1),0))-(vx1-pt(poly(poly,n),2),vy1-
pt(poly(poly,n),0))
        LINE(vx2-pt(poly(poly,n-1),2),vy2-
pt(poly(poly,n-1),1))-(vx2-pt(poly(poly,n),2),vy2-
pt(poly(poly,n),1))
        LINE(vx3+pt(poly(poly,n-1),0),vy3-
pt(poly(poly,n-1),1))-(vx3+pt(poly(poly,n),0),vy3-
pt(poly(poly,n),1))
      NEXT
      LINE(vx1-pt(poly(poly,n-1),2),vy1-
pt(poly(poly,n-1),0))-(vx1-pt(poly(poly,1),2),vy1-
pt(poly(poly,1),0))
      LINE(vx2-pt(poly(poly,n-1),2),vy2-
pt(poly(poly,n-1),1))-(vx2-pt(poly(poly,1),2),vy2-
pt(poly(poly,1),1))
      LINE(vx3+pt(poly(poly,n-1),0),vy3-
pt(poly(poly,n-1),1))-(vx3+pt(poly(poly,1),0),vy3-
pt(poly(poly,1),1))
    END IF
  END IF
RETURN

erase.poly:
  COLOR 0
  GOSUB drw.poly
  COLOR 1
RETURN

high.poly:
  COLOR 3
  GOSUB drw.poly
  COLOR 1
RETURN

unhigh.poly:
  GOSUB drw.poly
RETURN

high.clr:
  y2=INT(c/8)
  x2=c-y2*8
  LINE(192+x2*14,143+y2*7)-
 (205+x2*14,149+y2*7),3,b
 RETURN
```

```
unhigh.clr:
  y2=INT(c/8)
  x2=c-y2*8
  LINE(192+x2*14,143+y2*7)-
(205+x2*14,149+y2*7),0,b
RETURN

ptslider:
  IF x<197 THEN
    p=pt
    GOSUB unhigh.pt
    pt=pt-1
    IF pt<1 THEN pt=1
    p=pt
    GOSUB drw.ptnum
    GOSUB high.pt
    GOSUB nobut
  ELSEIF x>298 THEN
    p=pt
    GOSUB unhigh.pt
    pt=pt+1
    IF pt>maxpt THEN pt=maxpt
    p=pt
    GOSUB drw.ptnum
    GOSUB high.pt
    GOSUB nobut
  ELSEIF x>199 AND x<295 THEN
    p=pt
    GOSUB unhigh.pt
    pt=x-199
    p=pt
    GOSUB drw.ptnum
    GOSUB high.pt
    GOSUB nobut
  END IF
RETURN

polyslider:
  IF x<197 THEN
    c=polyclr(poly)
    GOSUB unhigh.clr
    GOSUB unhigh.poly
    poly=poly-1
    IF poly<1 THEN poly=1
    GOSUB drw.polynum
    GOSUB high.poly
    c=polyclr(poly)
    GOSUB high.clr
    GOSUB nobut
  ELSEIF x>298 THEN
    c=polyclr(poly)
    GOSUB unhigh.clr
    GOSUB unhigh.poly
    poly=poly+1
    IF poly>maxpoly THEN poly=maxpoly
    GOSUB drw.polynum
    GOSUB high.poly
    c=polyclr(poly)
    GOSUB high.clr
    GOSUB nobut
  ELSEIF x>199 AND x<295 THEN
    c=polyclr(poly)
    GOSUB unhigh.clr
    GOSUB unhigh.poly
    poly=x-199
    GOSUB drw.polynum
    GOSUB high.poly
    c=polyclr(poly)
    GOSUB high.clr
    GOSUB nobut
  END IF
RETURN

edact:
  IF x>4 AND x<122 AND y>150 AND y<158 THEN GOSUB
frmslider:RETURN
  IF x>40 AND x<250 AND y>136 AND y<148 THEN GOSUB
drw.frm:RETURN
  IF x>140 AND x<250 AND y>152 AND y<164 THEN GOSUB
drw.allfrm:RETURN
  IF x>260 AND x<306 AND y>136 AND y<148 THEN GOSUB
playbut:RETURN
  IF x>260 AND x<306 AND y>152 AND y<164 THEN GOSUB
stopbut:RETURN
  IF x>263 AND x<303 AND y>178 AND y<183 THEN GOSUB
spdslider:RETURN
  IF y>168 AND y<176 THEN
    IF x>92 AND x<140 THEN GOSUB mod.xrot:RETURN
    IF x>147 AND x<196 THEN GOSUB mod.yrot:RETURN
    IF x>203 AND x<252 THEN GOSUB mod.zrot:RETURN
  END IF
  IF y>177 AND y<185 THEN
    IF x>92 AND x<140 THEN GOSUB mod.xtran:RETURN
    IF x>147 AND x<196 THEN GOSUB mod.ytran:RETURN
    IF x>203 AND x<252 THEN GOSUB mod.ztran:RETURN
  END IF
RETURN

frmslider:
  IF x<12 THEN
    frm=frm-1
    IF frm<1 THEN frm=1
    GOSUB drw.frmnum
    GOSUB drw.update
    GOSUB drw.factors
    GOSUB putfrm
    GOSUB nobut
  ELSEIF x>114 THEN
    frm=frm+1
    IF frm>12 THEN frm=12
    GOSUB drw.frmnum
    GOSUB drw.update
    GOSUB drw.factors
    GOSUB putfrm
    GOSUB nobut
  ELSEIF x>16 AND x<110 THEN
    frm=INT((x-16)/8)+1
    GOSUB drw.frmnum
    GOSUB drw.update
    LINE(0,0)-(311,131),0,bf
    GOSUB drw.factors
    GOSUB putfrm
    GOSUB nobut
  END IF
RETURN

drw.frm:
  GOSUB high.redraw
  GOSUB drw.frame
  GOSUB getfrm
  frmchg(frm)=0
  GOSUB drw.update
  GOSUB unhigh.redraw
  GOSUB nobut
RETURN

drw.allfrm:
  GOSUB high.redraw2
  tfrm=frm
```

```
   FOR frm=1 TO 12
      GOSUB drw.frmnum
      GOSUB drw.factors
      GOSUB drw.frame
      GOSUB getfrm
      frmchg(frm)=0
      GOSUB drw.update
   NEXT
   frm=tfrm
   GOSUB drw.frmnum
   GOSUB drw.factors
   GOSUB putfrm
   GOSUB unhigh.redraw2
   GOSUB nobut
RETURN

playbut:
   GOSUB high.play
   GOSUB unhigh.stop
   GOSUB freeze.menu
   frminc=1
   clickstop=0
   WHILE NOT(clickstop)
      frm=frm+frminc
      IF frm>12 THEN
         IF actrev THEN
            frm=11
            frminc=(-1)
         ELSEIF actrpt THEN
            frm=1
         ELSE
            frm=1
            clickstop=(-1)
         END IF
      END IF
      IF frm<1 THEN
         IF actrpt THEN
            frm=2
            frminc=1
         ELSE
            frm=1
            clickstop=(-1)
         END IF
      END IF
      GOSUB putfrm
      GOSUB drw.frmnum
      FOR n=0 TO 39-spd
         b=MOUSE(0)
         x=MOUSE(1)
         y=MOUSE(2)
         IF b<>0 THEN
            IF x>260 AND x<306 AND y>152 AND y<164
THEN clickstop=(-1)
               n=39-spd
            END IF
            IF x>263 AND x<303 AND y>178 AND y<183
THEN
               spd=x-263
               GOSUB drw.spdnum
            END IF
         END IF
      NEXT
   WEND
   GOSUB drw.factors
   GOSUB drw.update
   GOSUB unhigh.play
   GOSUB high.stop
   GOSUB unfreeze.menu
   GOSUB nobut
RETURN
```



```
stopbut:
RETURN

spdslider:
   spd=x-263
   GOSUB drw.spdnum
RETURN

mod.xrot:
   s$=STR$(xrot(frm))
   xt=108:yt=175
   maxchar=4
   numonly=1
   GOSUB getstring2
   xrot(frm)=VAL(s$)
   xrot(frm)=xrot(frm) MOD 360
   IF xrot(frm)<0 THEN xrot(frm)=xrot(frm)+360
   GOSUB drw.factors
   frmchg(frm)=1
   GOSUB drw.update
   GOSUB nobut
RETURN

mod.yrot:
   s$=STR$(yrot(frm))
   xt=164:yt=175
   maxchar=4
   numonly=1
   GOSUB getstring2
   yrot(frm)=VAL(s$)
   yrot(frm)=yrot(frm) MOD 360
   IF yrot(frm)<0 THEN yrot(frm)=yrot(frm)+360
   GOSUB drw.factors
   frmchg(frm)=1
```

```
   GOSUB drw.update
   GOSUB nobut
RETURN

mod.zrot:
   s$=STR$(zrot(frm))
   xt=220:yt=175
   maxchar=4
   numonly=1
   GOSUB getstring2
   zrot(frm)=VAL(s$)
   zrot(frm)=zrot(frm) MOD 360
   IF zrot(frm)<0 THEN zrot(frm)=zrot(frm)+360
   GOSUB drw.factors
   frmchg(frm)=1
   GOSUB drw.update
   GOSUB nobut
RETURN

mod.xtran:
   s$=STR$(xtran(frm))
   xt=108:yt=184
   maxchar=4
   numonly=1
   GOSUB getstring2
   xtran(frm)=VAL(s$)
   IF xtran(frm)<-90 THEN xtran(frm)=(-90)
   IF xtran(frm)>90 THEN xtran(frm)=90
   GOSUB drw.factors
   frmchg(frm)=1
   GOSUB drw.update
   GOSUB nobut
RETURN

mod.ytran:
   s$=STR$(ytran(frm))
   xt=164:yt=184
   maxchar=3
   numonly=1
   GOSUB getstring2
   ytran(frm)=VAL(s$)
   IF ytran(frm)<-8 THEN ytran(frm)=(-8)
   IF ytran(frm)>8 THEN ytran(frm)=8
   GOSUB drw.factors
   frmchg(frm)=1
   GOSUB drw.update
   GOSUB nobut
RETURN

mod.ztran:
   s$=STR$(ztran(frm))
   xt=220:yt=184
   maxchar=4
   numonly=1
   GOSUB getstring2
   ztran(frm)=VAL(s$)
   IF ztran(frm)<0 THEN ztran(frm)=0
   IF ztran(frm)>999 THEN ztran(frm)=999
   GOSUB drw.factors
   frmchg(frm)=1
   GOSUB drw.update
   GOSUB nobut
RETURN

drw.frame:
   GOSUB unit.matrix
   IF xrot(frm)>0 THEN GOSUB apply.xrot
   IF yrot(frm)>0 THEN GOSUB apply.yrot
   IF zrot(frm)>0 THEN GOSUB apply.zrot
   IF ztran(frm)>0 THEN GOSUB apply.ztran
   GOSUB transform.pts
   GOSUB convert2scr
   GOSUB sort.poly
   LINE(0,0)-(311,131),0,bf
   GOSUB drw.object
RETURN

matprep:     'bit of matrix preparation
   FOR row=0 TO 3
      FOR col=0 TO 3
         tr1(row,col)=tran(row,col)
         tr2(row,col)=0
      NEXT
   NEXT
RETURN

unit.matrix:     'create a unit matrix
   FOR row=0 TO 3
      FOR col=0 TO 3
         tran(row,col)=0
         IF row=col THEN tran(row,col)=1
      NEXT
   NEXT
RETURN

matmult:     'multiply transformation matricies
   FOR row=0 TO 3
      FOR col=0 TO 3
         t=0
         FOR el=0 TO 3
            t=t+tr1(row,el)*tr2(el,col)
         NEXT
         tran(row,col)=t
      NEXT
   NEXT
RETURN

apply.xrot:     'rotate object about x axis
   GOSUB matprep
   rad=xrot(frm)*3.1416/180
   tr2(0,0)=1:tr2(3,3)=1
   tr2(1,1)=COS(rad):tr2(1,2)=SIN(rad)*(-1)
   tr2(2,1)=SIN(rad):tr2(2,2)=COS(rad)
   GOSUB matmult
RETURN

apply.yrot:     'rotate object about y axis
   GOSUB matprep
   rad=yrot(frm)*3.1416/180
   tr2(1,1)=1:tr2(3,3)=1
   tr2(0,0)=COS(rad):tr2(0,2)=SIN(rad)
   tr2(2,0)=SIN(rad)*(-1):tr2(2,2)=COS(rad)
   GOSUB matmult
RETURN

apply.zrot:     'rotate object about z axis
   GOSUB matprep
   rad=zrot(frm)*3.1416/180
   tr2(2,2)=1:tr2(3,3)=1
   tr2(0,0)=COS(rad):tr2(0,1)=SIN(rad)*(-1)
   tr2(1,0)=SIN(rad):tr2(1,1)=COS(rad)
   GOSUB matmult
RETURN

apply.ztran:     'translate object along z axis
   GOSUB matprep
   tr2(0,0)=1:tr2(1,1)=1
   tr2(2,2)=1:tr2(3,3)=1
   tr2(3,2)=ztran(frm)
   GOSUB matmult
RETURN
```

```
transform.pts:
  COLOR 1,0
  FOR p=1 TO 95
    LOCATE 1,5
    PRINT"Calculating Point";p
    IF pt(p,0)<>0 OR pt(p,1)<>0 OR pt(p,2)<>0 THEN
      FOR col=0 TO 3
        t=0
        FOR el=0 TO 3
          t=t+pt(p,el)*tran(el,col)
        NEXT
        tpt(p,col)=t
      NEXT
    END IF
  NEXT
RETURN

convert2scr:     'convert points to screen
coordinates
  FOR p=1 TO 95
    r=zeye/(tpt(p,2)+zeye)
    tpt(p,0)=INT(tpt(p,0)*r)+hoff
    tpt(p,1)=(INT(tpt(p,1)*r))*(-1)+voff
  NEXT
RETURN

reset.polyorder:
  FOR n=1 TO maxpoly
    polyord(n,0)=0
    polyord(n,1)=0
  NEXT
RETURN

sort.poly:
  GOSUB reset.polyorder
  'find average of all polygons' z coord
  FOR n=1 TO maxpoly
    LOCATE 1,17
    PRINT"Polygon"n
    IF vrt(n)>0 THEN
      t=0
      FOR n2=1 TO vrt(n)
        t=t+tpt(poly(n,n2),2)
      NEXT
      polyord(n,0)=INT(t/vrt(n))
    END IF
  NEXT
  'sort polygons by average z coord
  FOR n=1 TO maxpoly
    LOCATE 1,24
    PRINT n;"again."
    IF vrt(n)>0 THEN
      t=(-100)
      p=(-1)
      FOR n2=1 TO maxpoly
        IF vrt(n2)>0 THEN
          IF polyord(n2,0)>t THEN
t=polyord(n2,0):p=n2
        END IF
      NEXT
      polyord(n,1)=p
      polyord(p,0)=(-100)
    END IF
  NEXT
  COLOR 1,2
RETURN

drw.object:
  FOR n=1 TO maxpoly
    IF vrt(n)>2 THEN
      FOR n2=1 TO vrt(polyord(n,1))
        AREA(tpt(poly(polyord(n,1),n2),0)+xtran
(frm),tpt(poly(polyord(n,1),n2),1)+ytran(frm))
      NEXT
      COLOR polyclr(polyord(n,1))
      AREAFILL
    END IF
  NEXT
  COLOR 1,2
RETURN

getfrm:
  GET(hoff-64+xtran(frm),voff-58+ytran(frm))-
(hoff+63+xtran(frm),voff+57+ytran(frm)),frame&((fr
-1)*frmsize)
RETURN

putfrm:
  PUT(hoff-64+xtran(frm),voff-
58+ytran(frm)),frame&((frm-1)*frmsize),PSET
RETURN

getstring:
  'enter here if no default string
  s$=""
getstring2:
  'enter here if have a default string
  GOSUB freeze.menu
  GOSUB nokey
  numchar=LEN(s$)
  CALL move&(rp&,xt,yt)
  PRINT s$;
```

```
  z$=""
getstring3:
    LINE(xt+numchar*8+1,yt-7)-
(xt+numchar*8+4,yt+2),3,bf
    z$=INPUT$(1)
    LINE(xt+numchar*8+1,yt-7)-
(xt+numchar*8+4,yt+2),2,bf
    IF z$=CHR$(8) OR z$=CHR$(31) OR z$=CHR$(127)
THEN
      IF numchar>0 THEN
        PRINT CHR$(8);" ";CHR$(8);
        numchar=numchar-1
        s$=LEFT$(s$,numchar)
      END IF
    END IF
    IF ASC(z$)>31 AND numchar<maxchar AND numonly=0
THEN
        s$=s$+z$
        PRINT z$;
        numchar=numchar+1
    END IF
    IF ASC(z$)>31 AND numchar<maxchar THEN
      IF numonly=1 THEN
        IF(z$>="0" AND z$<="9")OR z$="-" THEN
          s$=s$+z$
          PRINT z$;
          numchar=numchar+1
        END IF
      END IF
    END IF
  IF z$<>CHR$(13) THEN getstring3
  numonly=0
  GOSUB unfreeze.menu
RETURN

click.continue:
  LOCATE 23,4
  PRINT"(Press mouse button to continue)";
  GOSUB nobut
  b=MOUSE(0)
  WHILE b=0
    b=MOUSE(0)
  WEND
RETURN

nobut:
  b=MOUSE(0)
  WHILE b<>0
    b=MOUSE(0)
  WEND
RETURN

nokey:
  z$=INKEY$
  WHILE z$<>""
    z$=INKEY$
  WEND
RETURN

init:
  DEFINT a-p,u-z
  DECLARE FUNCTION setdrmd LIBRARY
  DECLARE FUNCTION move LIBRARY
  LIBRARY "graphics.library"
  SCREEN 1,320,200,5,1
  WINDOW 2,"ROT",(0,0)-(311,186),0,1
  WINDOW OUTPUT 2
  rp&=WINDOW(8) 'pointer to raster port
  PALETTE 0,0,0,0
  PALETTE 2,0,.5,0
  PALETTE 31,0,.25,0
  PALETTE 30,.7,.7,0
  LOCATE 3,1
  GOSUB intro
  DIM pt(95,3),poly(95,6),polyclr(95),vrt(95)
  DIM xrot(12),yrot(12),zrot(12)
  DIM xtran(12),ytran(12),ztran(12),frmchg(12)
  DIM
tran(3,3),tr1(3,3),tr2(3,3),tpt(95,3),polyord(95,1)

  DIM frame&(27876)
  pt=1:poly=1
  objscr=-1:actscr=0
  vx1=43:vx2=43:vx3=136
  vy1=43:vy2=135:vy3=135
  maxpt=95:maxpoly=95
  frm=1:spd=20
  frmsize=2323
  hoff=156:voff=66:zeye=440
  actrpt=0:actrev=0
  FOR n=1 TO maxpt
    pt(n,3)=1
  NEXT
  GOSUB init.menu
  COLOR 1,0
  GOSUB click.continue
  COLOR 1,2
  GOSUB drw.objscr
RETURN

init.menu:
  MENU 1,0,1,"ROT"
  MENU 1,1,1," Files "
  MENU 1,2,0,"-------"
  MENU 1,3,1," Quit  "
  MENU 2,0,1,"Object"
  MENU 2,1,2,"  Object Editor "
  MENU 2,2,1,"  Load Object   "
  MENU 2,3,1,"  Save Object   "
  MENU 2,4,1,"  New Object    "
  MENU 3,0,1,"Action"
  MENU 3,1,1,"  Action Editor  "
  MENU 3,2,0,"  Load Action    "
  MENU 3,3,0,"  Save Action    "
  MENU 3,4,0,"  New Action     "
  MENU 3,5,0,"----------------"
  MENU 3,6,0,"  Repeat at end  "
  MENU 3,7,0,"  Reverse at end "
  MENU 3,8,0,"  Calc between..."
  MENU 4,0,0," "
  MENU 4,1,0," "
RETURN

freeze.menu:
  MENU 1,0,0
  MENU 2,0,0
  MENU 3,0,0
RETURN

unfreeze.menu:
  MENU 1,0,1
  MENU 2,0,1
  MENU 3,0,1
RETURN

cleanup:
  WINDOW CLOSE 2
  SCREEN CLOSE 1
  LIBRARY CLOSE
  PALETTE 0,0,.25,.55
```

```
   PALETTE 2,0,0,0
   MENU RESET
RETURN

drw.objscr:
   LINE(0,0)-(320,200),2,bf
   LINE(2,2)-(83,83),0,bf
   LINE(2,95)-(83,176),0,bf
   LINE(96,95)-(177,176),0,bf
   LINE(186,2)-(310,53),1,b
   LINE(186,60)-(310,176),1,b
   LINE(191,142)-(304,171),0,bf
   FOR y=0 TO 3
      FOR x=0 TO 7
         LINE(193+x*14,144+y*7)-
(204+x*14,148+y*7),y*8+x,bf
      NEXT
   NEXT
   x=189:y=126:GOSUB drw.button
   y=110:GOSUB drw.button
   y=94:GOSUB drw.button
   y=36:GOSUB drw.button
   y=24:GOSUB drw.scroll
   y=82:GOSUB drw.scroll
   CALL move&(rp&,228,12):PRINT"POINT"
   CALL move&(rp&,236,21):PRINT"#"
   CALL move&(rp&,208,45):PRINT"Zero Point"
   CALL move&(rp&,220,70):PRINT"POLYGON"
   CALL move&(rp&,236,79):PRINT"#"
   CALL move&(rp&,200,103):PRINT"Add above Pt"
   CALL move&(rp&,200,119):PRINT"Undo last Pt"
   CALL move&(rp&,192,135):PRINT"Delete Polygon"
   CALL move&(rp&,10,92):PRINT"Z   TOP"
   CALL move&(rp&,10,185):PRINT"Z   SIDE";
   CALL move&(rp&,107,185):PRINT"FRONT   X";
   CALL move&(rp&,86,16):PRINT"X"
   CALL move&(rp&,86,109):PRINT"Y"
   x=2:y=89:GOSUB drw.leftarrow
   y=182:GOSUB drw.leftarrow
   x=89:y=2:GOSUB drw.uparrow
   y=95:GOSUB drw.uparrow
   x=177:y=182:GOSUB drw.rightarrow
   GOSUB drw.ptnum
   GOSUB drw.polynum
   GOSUB drw.views
   c=polyclr(poly)
   GOSUB high.clr
   GOSUB nobut
RETURN

drw.button:
   LINE(x,y)-(x+116,y+12),1,b
   LINE(x+2,y+13)-(x+117,y+13),0
   LINE -(x+117,y+1),0
RETURN

drw.scroll:
   LINE(x,y)-(x+116,y+8),1,b
   LINE(x+8,y)-(x+109,y+8),1,b
   LINE(x+2,y+9)-(x+117,y+9),0
   LINE -(x+117,y+1),0
   COLOR 0
   AREA(x+3,y+4):AREA(x+5,y+2):AREA(x+5,y+6)
   AREAFILL
   AREA(x+112,y+2):AREA(x+112,y+6):AREA(x+114,y+4)
   AREAFILL
   COLOR 1,2
RETURN
```



```
drw.leftarrow:
   AREA(x,y):AREA(x+3,y-3):AREA(x+3,y+3):AREAFILL
   LINE(x,y)-(x+6,y)
RETURN

drw.rightarrow:
   AREA(x,y):AREA(x-3,y-3):AREA(x-3,y+3):AREAFILL
   LINE(x,y)-(x-6,y)
RETURN

drw.uparrow:
   AREA(x,y):AREA(x-3,y+3):AREA(x+3,y+3):AREAFILL
   LINE(x,y)-(x,y+6)
RETURN

drw.ptnum:
   LINE(199,26)-(296,30),0,bf
   LINE(198+pt,26)-(201+pt,30),3,bf
   CALL move&(rp&,244,21)
   PRINT RIGHT$("00"+STR$(pt),2)
RETURN

drw.polynum:
   LINE(199,84)-(296,88),0,bf
   LINE(198+poly,84)-(201+poly,88),3,bf
   CALL move&(rp&,244,79)
   PRINT RIGHT$("00"+STR$(poly),2)
RETURN

drw.actscr:
   LINE(0,0)-(311,131),0,bf
   LINE(0,132)-(311,186),2,bf
   x=140:y=136:GOSUB drw.button2
   y=152:GOSUB drw.button2
```

```
   x=260:y=136:GOSUB drw.button3
   y=152:GOSUB drw.button3
   LINE(260,178)-(306,183),1,b
   LINE(262,184)-(307,184),0
   LINE -(307,179),0
   LINE(4,150)-(122,158),1,b
   LINE(12,150)-(114,158),1,b
   LINE(6,159)-(123,159),0
   LINE -(123,151),0
   COLOR 0
   AREA(7,154):AREA(9,152):AREA(9,156):AREAFILL
   AREA(117,152):AREA(119,154):AREA(117,156)
   AREAFILL
   COLOR 1,2
   CALL move&(rp&,28,145):PRINT"FRAME #"
   CALL move&(rp&,4,175):PRINT"Rotations: X=      Y=
    Z="
   CALL move&(rp&,4,184):PRINT"Translate: X=      Y=
    Z=";
   GOSUB unhigh.redraw
   GOSUB unhigh.redraw2
   GOSUB unhigh.play
   GOSUB high.stop
   CALL move&(rp&,264,175):PRINT"Speed"
   GOSUB drw.frmnum
   GOSUB drw.spdnum
   GOSUB drw.update
   GOSUB drw.factors
   GOSUB putfrm
RETURN

drw.frmnum:
   LINE(15,152)-(111,156),0,bf
   LINE(7+frm*8,152)-(15+frm*8,156),3,bf
   CALL move&(rp&,84,145)
   PRINT RIGHT$("00"+STR$(frm),2)
RETURN

drw.spdnum:
   LINE(262,180)-(304,181),0,b
   LINE(261+spd,180)-(265+spd,181),3,b
RETURN

drw.update:
   LINE(115,138)-(131,146),2,bf
   IF frmchg(frm)<>0 THEN
      LINE(115,142)-(131,142),3
      LINE -(127,138),3
      LINE(131,142)-(127,146),3
   END IF
RETURN

drw.factors:
   CALL move&(rp&,108,175)
   PRINT LEFT$(STR$(xrot(frm))+"    ",4)
   CALL move&(rp&,164,175)
   PRINT LEFT$(STR$(yrot(frm))+"    ",4)
   CALL move&(rp&,220,175)
   PRINT LEFT$(STR$(zrot(frm))+"    ",4)
   CALL move&(rp&,108,184)
   PRINT LEFT$(STR$(xtran(frm))+"    ",4);
   CALL move&(rp&,164,184)
   PRINT LEFT$(STR$(ytran(frm))+"    ",4);
   CALL move&(rp&,220,184)
   PRINT LEFT$(STR$(ztran(frm))+"    ",4);
RETURN

unhigh.redraw:
   LINE(141,137)-(249,147),2,bf
   CALL move&(rp&,148,145)
   PRINT"Redraw Frame"
RETURN

high.redraw:
   LINE(141,137)-(249,147),3,bf
   COLOR 1,3
   CALL move&(rp&,148,145)
   PRINT"Redraw Frame"
   COLOR 1,2
RETURN

unhigh.redraw2:
   LINE(141,153)-(249,163),2,bf
   CALL move&(rp&,156,161)
   PRINT"Redraw All"
RETURN

high.redraw2:
   LINE(141,153)-(249,163),3,bf
   COLOR 1,3
   CALL move&(rp&,156,161)
   PRINT"Redraw All"
   COLOR 1,2
RETURN

unhigh.play:
   LINE(261,137)-(305,147),2,bf
   CALL move&(rp&,268,145)
   PRINT"Play"
RETURN

high.play:
   LINE(261,137)-(305,147),3,bf
   COLOR 1,3
   CALL move&(rp&,268,145)
   PRINT"Play"
   COLOR 1,2
RETURN

unhigh.stop:
   LINE(261,153)-(305,163),2,bf
   CALL move&(rp&,268,161)
   PRINT"Stop"
RETURN

high.stop:
   LINE(261,153)-(305,163),3,bf
   COLOR 1,3
   CALL move&(rp&,268,161)
   PRINT"Stop"
   COLOR 1,2
RETURN

drw.button2:
   LINE(x,y)-(x+110,y+12),1,b
   LINE(x+2,y+13)-(x+111,y+13),0
   LINE -(x+111,y+1),0
RETURN

drw.button3:
   LINE(x,y)-(x+46,y+12),1,b
   LINE(x+2,y+13)-(x+47,y+13),0
   LINE -(x+47,y+1),0
RETURN
```

•AC•

By Popular Demand!

Ok, Ok! We give in. Our readers have requested a subscription form and we are giving in. The small coupon in the corner is a direct result of the dozens of calls and letters from prospective subscribers. In all honesty, we have never put a subscription form in the magazine, because we wanted your thoughts and ideas by letter.

However, if there are some Amiga users out there who do not want to drop us a line and tell us of the Amazing things they are doing, well, just fill out the card below with check or money order for $24.00 U.S., $30.00 Canada and Mexico, or $35.00 overseas (first class and airmail rates are available upon request) and send to:

PiM Publications Inc.
P.O. Box 869
Fall River, MA 02722

We will start your subscription request with our next available issue.

If you need any of our first six **Back Issues**, they are available at $4.00 each and can be ordered at the same time as your subscription.

While sending the coupon, please don't forget our long list of **Public Domain Software** listed in the back of this issue!

So, there you have a quick, easy way to fully support your Amiga. But, if you are not in such a hurry, we would still like to see your thoughts, dreams, hopes, and ideas in your letters.

Thank you.

Amazing Computing

Yes! I am taking the easy way, please start my subscription with the next available issue! I have enclosed $24.00 for 12 issues U.S.($30.00 Canada and Mexico, $35.00 overseas).

_____
name

_____
street

_____, _____, _____
city state zip code

I am mailing this to:
PiM Publications, Inc.
P.O. Box 869
Fall River, MA 02722

Thanks for your order.

Computer West

(602) 849-4795

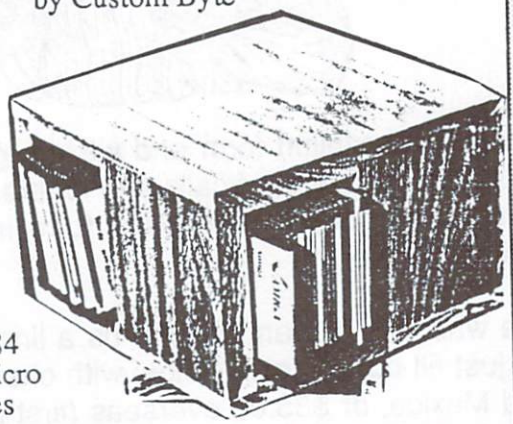
4130 N. 75th Ave. #105
Phoenix, AZ 85033

DEALER INQUIRES WELCOME

NEW Amiga® Products!

Solid Oak Swivel Custom Disk Cabinet

by Custom Byte



Holds 84 3.5" Micro Floppies

Put all your disks in one place - at your fingertips

$49.95

Gallery of Images For DELUXE PRINT®

Chapter 1

Featuring:
States, Social Activities, Southwestern Art, Mix and Match

only **$19.95**

MEMORY EXPANSION BOARD

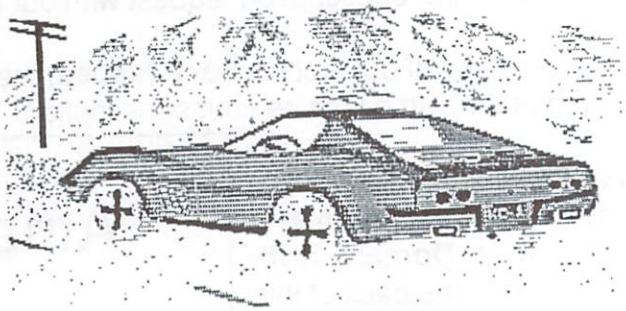
A BIG **2** mb.

Features:
- Speed
- Auto configuring
- Mounts on expansion port
- Zero wait states

A must for business, graphics, database, and cad programs. **$795.00**

Computer Art Gallery of Animations

by Ken Costello



Corvette example; wheels spin, telephone poles flash by and road moves. 23 animations, Explore the animations of your computer.

$19.95

☐ Art Disk Deluxe Print	19.95	☐ Memory Board	$795.00
☐ Art Disk Deluxe Paint	19.95	☐ Oak disk holder	$69.95

☐ check/money order ☐ Credit Card No._____

Exp. date_____ Sign._____

☐ Visa ☐ M/C ☐ Amer. Exp.

$3.00 shipping and handling
AZ residents add 6.5% sales tax
(602) 849-4795

Mail to: Computer West
4130 N. 75th Ave. #105
Phoenix, AZ 85033

Name_____
Address_____
City_____ State____ Zip_____

Forth!

By Jon Bryan

Last month I began the planning stages of an application which would simulate a bouncing ball in three dimensions. Some of the necessary equations were presented, along with a discussion of the many choices and compromises which must be made in writing such an application. I also made the decision to use "Multi-Forth" from Creative Solutions, Inc. as the language implementation, largely because it was (and still is at the time this is being written) the most powerful version yet to be released.

This month I'm finally including some actual code which you can run if you have Multi-Forth. It's an implementation of the Bresenham/Michener circle algorithm as published in "Fundamentals of Interactive Computer Graphics" by J.D. Foley and A. Van Dam. It may not seem at first glance to have much to do with bouncing balls, but I ask you to bear with me. I think it will prove to be useful. Let me explain my reason for implementing the algorithm. I would like to draw a series of shaded spheres to give the perspective view of the ball as it travels from the front to the back of the "room." In order to derive the x and y coordinates for each pixel which constitutes a point on the surface of the sphere, the radius of the sphere must be calculated for each horizontal "slice" (raster scan line) through the sphere. Bresenham's algorithm provides a convenient method for calculating that radius quickly and efficiently. For each slice, the radius provides a starting and an ending pixel relative to the central axis of the sphere. The x and y pixel coordinates and the radius of the sphere may then be used to calculate the amount of illumination reflected to your eye from that point on the surface of the sphere.

The algorithm presented here will actually draw circles; something which won't be required for bouncing balls. For calculating the illumination of the sphere we'll be more interested in the raw numbers -- the coordinates of the pixels, rather than the pixels themselves. But since pictures are easier to understand than numbers, and because it provides some instant gratification, I decided to draw circles now and save the pure calculations for later.

Even if I later decide to take a different tack, this month's algorithm is worthwhile for its own sake. It provides an excellent example of the use of local variables, and it is also slightly faster than the circle-drawing algorithm included in Multi-Forth.

At this point I must admit a weakness on my part. I don't yet know how to create animateable objects on the Amiga from Forth. What I expect to use are Blitter Objects, or "BOBs," as they will provide the greatest flexibility and also allow a greater number of colors than hardware sprites. The extra colors will be required for the different levels of illumination. If there were no background to contend with the balls could be animated by brute force, using the Blitter to move a chunk of the screen from one place to another, and to move a new image into place as the ball changed its apparent size. But, since we must preserve the background over which the ball is animated, BOBs seem to be the obvious way to go. I just don't know how to use them yet.

STACKS AND LOCALS

Forth programmers are accustomed to handling most data on the stack, but Multi-Forth provides an alternative in the form of "local" variables. Besides SWAP, ROT, DUP, OVER and all the other stack operators, in Multi-Forth you can opt for named, local variables within words to streamline definitions and make your code more readable. After only a few weeks of using them I've been converted. Local variables are convenient, readable, often faster than using the stack, and they obviate one of the objections users of more traditional languages have had to Forth in the past. There may be an ultra-conservative Forth programmer somewhere who will object to local variables, but I think it's a great idea whose time has come.

One component of the circle-drawing algorithm provides an excellent example. The algorithm calculates one octant (45 degrees) of the circle, and uses symmetry to plot the other seven pixels. The procedure which plots the eight points, named CIRCLE_POINTS, is a perfect choice for the use of local variables.

Referring to the listing, CIRCLE_POINTS expects the x and y coordinates relative to the origin on the stack. OVER NEGATE OVER NEGATE produces -x and -y, and LOCALS| -y -x y x | then creates the variables, assigning values to them in the order that the values come off the stack. Use of the local variable within the definition causes its value to be pushed onto the stack to be used by DOT, a word included in Multi-Forth which plots a pixel in the current window. The eight invocations of DOT then use symmetry to plot the eight pixels.

The words ADDR.OF and TO allow modification of the value of a local variable. "ADDR.OF d" places the address of d on the stack, and "0 TO d" would set the local variable d to zero. It's important to realize that variables created by LOCALS| are transient. They only exist within the word in which they're used, so the same variable name may be used within any number of words and mean something different each time, though I wouldn't recommend the practice.

The word which calculates the x and y coordinates for the circle is named MICH_CIRCLE after J. Michener, who derived the algorithm from one that J. Bresenham developed for pen plotters. A reasonably detailed treatment of the algorithm may be found in the text "Fundamentals of Interactive Computer Graphics" referenced earlier.

I'll try to distill the essence of the algorithm. MICH_CIRCLE expects the x and y coordinates of the origin of the circle and the radius on the stack. First it initializes a "decision variable" d. Then, within a loop, a new value for the decision variable is calculated incrementally and the sign of the new value is used to determine which pixel is closest to the radius of the true circle. The x coordinate of the circle is incremented on each pass through the loop and the value of the y coordinate is decremented if the decision variable is greater than or equal to zero. The loop is exited when x>=y. Refer to the text if you wish to study the algebraic manipulations from which the calculations used in the algorithm are derived.

CODE OPTIMIZATION

Code optimization is a dangerous subject, in more ways than one. The quest for ultimate efficiency can be a frustrating one, with many pitfalls. I fell into one myself and thought I should share it with you. After my success at coding MICH_CIRCLE and my pleasure at seeing it actually WORK I thought I'd have a go at speeding it up a bit. Multi-Forth, like most Forths, includes an assembler which can be used for optimization. At times it's indispensable, but at other times overzealous use of it can result in a lot of wasted time. In this particular case my use of it turned out to be mostly wasted effort.

Typically, if you want an application to run faster you translate the inner loops into machine code, since you may be spending 90% of your time executing 10% of the code. I translated the portion of the algorithm which recalculated the decision variable and the values of x and y, naming it DECIDE.XY, made the substitution in MICH_CIRCLE2, and lo and behold, I got a whopping one-half of one percent improvement in the execution speed! At this point I realized that the effort was pointless, but I still couldn't resist tweaking things a bit more. MICH_CIRCLE2 uses locals to pass the parameters to DECIDE.XY, then updates those variables upon exiting DECIDE.XY by using the Multi-Forth word TO. It was obvious that what little I gained by using machine code was being reabsorbed in manipulating the variables.

In MICH_CIRCLE3 I rearranged things so that the decision variable and the x and y coordinates were left on the stack, reducing the overhead for DECIDE.XY. Now when I ran this "improved" version the speed increased by all of 3%! You have undoubtedly realized by now that Multi-Forth is already quite fast, and more importantly, that I was optimizing the wrong part of the code. Unable to content myself with the obvious truth, I decided to try ONE MORE THING, and there at least I accomplished something, even if it wasn't what I expected.

PROBLEMS WITH THE ASSEMBLER

Multi-Forth has an interesting feature which allows assembly code to be included "in line." By using the words >CODE and >FORTH one can insert machine instructions right into the middle of a high-level definition, and that's what I did in MICH_CIRCLE4. When I tried it I got an "empty stack" error, and for the life of me I couldn't understand why. I spent an hour or so experimenting and scratching my head (I even called Creative Solutions), and eventually tracked the problem to a mistake in the Assembler. By the time you read this the problem will be corrected on the latest release, but if you're not sure, go into the "assembler" file using your favorite editor and find >FORTH. You'll see the phrase "HERE 10 +" which should be changed to read "HERE 0A +" and you're all set. The problem was that HERE 10 + created an offset for a Load Effective Address instruction which was too large and caused a branch to parts unknown.

A FEW FINAL NOTES

On a 640x200 display MICH_CIRCLE doesn't actually draw circles. It draws tall, skinny ellipses, because the aspect ratio isn't 1:1. Executing "220 100 XYSCALE" will cause it to draw something much closer to a circle. Before you can compile the code you will have to "include assembler" for the CODE definitions. Multi-Forth doesn't include the assembler by default. Those who are familiar with standard assembler will probably find Forth assembly language confusing, since it's "Reverse Polish" just like Forth. A more serious problem is that each Forth vendor's assembler is slightly different, making translation more difficult. Some Forth vendors have decided to implement the standard Motorola assembler syntax for just that reason.

The next step is to execute "samplewindow" to open a window in which the circles can be plotted.

The benchmark I used for timing is as follows:

```
: TEST
  100 100 100 0
  DO 2DUP I MICH_CIRCLE LOOP
  2DROP ;
```

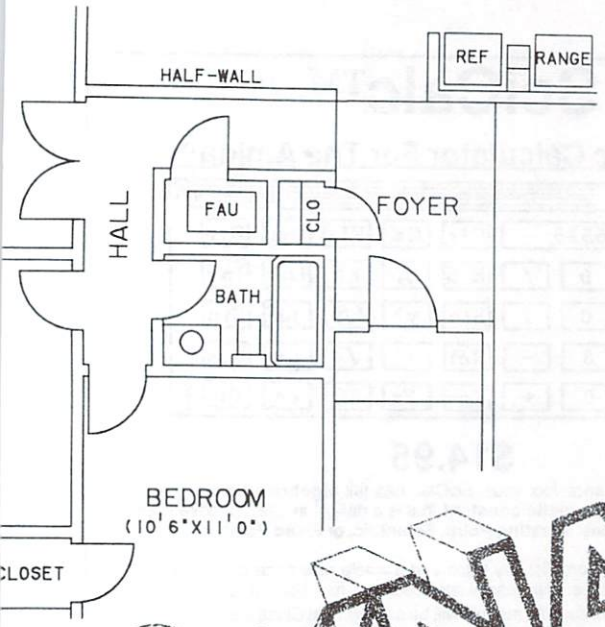
which draws 100 circles at a common origin with radii from 0 to 99.

SOURCE LISTING FOLLOWS:

```
\ Bresenham/Michener circle algorithm.
\ From "Fundamentals of Interactive Computer
\ Graphics"
\ by Foley and Van Dam.
\ Jon R. Bryan 6-24-86

\ This routine plots eight pixels of the circle
  using symmetry

: CIRCLE_POINTS    ( x\y -- )
   OVER  NEGATE  OVER  NEGATE
   LOCALS|   -y   -x   y   x   |
   x y  DOT    y x  DOT   -x y  DOT    y -x  DOT
   x -y  DOT   -y x  DOT  -x -y  DOT  -y -x  DOT ;

\ First we do everything in high level

: MICH_CIRCLE    ( xcenter\ycenter\radius -- )
   3  OVER  2*- ( decision variable d=3-2*radius )
   0  ( the initial value of x )
   GET.XYOFFSET  ( DOT is relative to the current
                   XYOFFSET )
   ( initially y=radius )
   LOCALS|  yoffset   xoffset   x   d   y   ycenter
 xcenter  |
 ( next set XYOFFSET to the origin of the circle )
   xoffset  xcenter  +   yoffset  ycenter  +
 XYOFFSET
```

```
   BEGIN     x  y  <
   WHILE     x  y  CIRCLE_POINTS
     d  0<
     IF    x  4*  6+    ADDR.OF  d  +! ( d=d+4*x+6)
     ELSE
       x  y  -  4*  10+ ADDR.OF  d  +!
        ( d=d+4*[x-y]+10 )
       -1  ADDR.OF  y  +!  ( decrement y )
     THEN
     1  ADDR.OF  x  +!  ( increment x )
   REPEAT
   x  y  = ( the pixel precisely on the diagonal )
   IF    x  y  CIRCLE_POINTS    THEN
   xoffset  yoffset  XYOFFSET  ;

\ Now we decide to get tricky and optimize
\ This code replaces the IF-ELSE-THEN section
  above

CODE DECIDE.XY    ( d\x\y -- d'\x'\y' )
   SP )+  REGLIST D0-D2  LONG MOVEM,
      ( move values to registers)
   D1  D3  LONG MOVE,  ( make a working copy of x)
   D2  LONG TST,   ( d<0? )
   MI  IF,    D3  02 #  LONG ASL,  D3 06 LONG ADDQ,
              D3  D2  LONG ADD,   ( d=4*x+6 )
      ELSE,   D0  D3  LONG SUB, D3 02 #  LONG ASL,
              D3  10  LONG ADDI,
              D3  D2  LONG ADD,   ( d=4*[x-y]+10 )
              D0  01  LONG SUBQ,  ( decrement y)
      THEN,   D1  01  LONG ADDQ,  ( increment x)
   REGLIST D0-D2  SP -)  LONG MOVEM,
     ( new stack values )
   NEXT   END-CODE

\ And we insert the new routine...

: MICH_CIRCLE2    ( xcenter\ycenter\radius -- )
   3  OVER  2*  -   0   GET.XYOFFSET
   LOCALS|  yoffset   xoffset   x   d   y   ycenter
 xcenter  |
   xoffset xcenter + yoffset  ycenter +   XYOFFSET
   BEGIN    x  y  <
   WHILE    x  y  CIRCLE_POINTS
            d  x  y  DECIDE.XY  TO  y  TO  x  TO  d
   REPEAT
   x  y  =
   IF    x  y  CIRCLE_POINTS    THEN
   xoffset  yoffset  XYOFFSET ;

\ Lo and behold, it makes hardly any difference
  (less than .5%)!
\ So we decide to try a slightly different tack.

: MICH_CIRCLE3    ( xcenter\ycenter\radius -- )
   GET.XYOFFSET
   LOCALS|   yoffset  xoffset | >R  ( save radius)
   SWAP  xoffset  +  SWAP  yoffset  +  XYOFFSET
   3  R@  2*  -    ( initialize decision variable )
   0  ( x)    R> ( now we have d\x\y on the stack )
   BEGIN    2DUP  <
   WHILE    2DUP  CIRCLE_POINTS    DECIDE.XY
   REPEAT
   2DUP  =
   IF    CIRCLE_POINTS  DROP    ELSE    2DROP  DROP
 THEN xoffset  yoffset  XYOFFSET ;

\ And find that we've shaved less than 3% off the
\ time! Which just goes to show that Multi-Forth is
\ quite fast to begin with, and that much more time
\ is spent plotting the points than calculating
\ their coordinates. But just to demonstrate one
\ more technique, here is an example of the use of
\ >CODE and >FORTH for in-line code.

: MICH_CIRCLE4    ( xcenter\ycenter\radius -- )
   GET.XYOFFSET
   LOCALS|   yoffset   xoffset | >R  ( save radius)
   SWAP  xoffset  +  SWAP  yoffset  +  XYOFFSET
   3  R@  2*  -   ( initial value of decision
                    variable )
   0  ( x)    R> ( y=radius initially )
   BEGIN    2DUP  <
   WHILE    2DUP  CIRCLE_POINTS
     >CODE
   SP  )+ REGLIST D0-D2 LONG MOVEM,
     D1 D3 LONG  MOVE,
     D2  LONG  TST,
     MI  IF,      D3  02  #  LONG  ASL,
                  D3  06  LONG  ADDQ,
                  D3  D2  LONG  ADD,
          ELSE,   D0  D3  LONG  SUB,
                  D3  02  #  LONG  ASL,
                  D3  10  LONG  ADDI,
                  D3  D2  LONG  ADD,
                  D0  01  LONG  SUBQ,
          THEN,   D1  01  LONG  ADDQ,
     REGLIST  D0-D2  SP  -)  LONG  MOVEM,
     >FORTH
   REPEAT
   2DUP  =
   IF    CIRCLE_POINTS  DROP    ELSE    2DROP  DROP
 THEN xoffset  yoffset  XYOFFSET;
```


•AC•



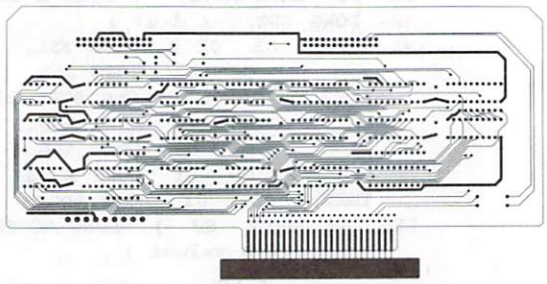
Finally, a proven CAD system for the Amiga

FROM MICRO-ILLUSIONS

DYNAMIC-CAD

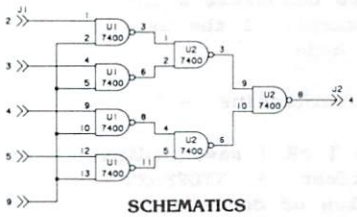
What AutoCAD* can do for the IBM*, Dynamic CAD can do with the Amiga... and a great deal more for less than a fourth of the price!

PRINTED CIRCUIT ARTWORK

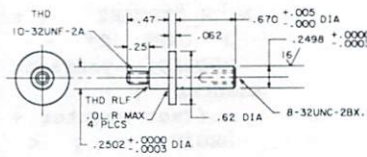


Emerging from years of successful problem solving applications in piping, and electronics for the aerospace industry, DYNAMIC CAD has brought a highly advanced and powerful CAD system together with today's most dynamic and versatile micro-computer, the Amiga. DYNAMIC CAD takes full advantage of Amiga's extensive capabilities with color, multiple modes of resolution, mouse functions, and easily accessible pull-down menus.

This is not some promised "vapor-ware." DYNAMIC CAD exists now and comes to the Amiga with a proven track record. The time and money-saving applications of DYNAMIC CAD for engineers and architects are truly astounding. Here is an advanced, 2-D drafting system with isometric capabilities that can be combined with many models of printers, plotters, and digitizers. In getting started you'll have the support of an extensive manual written in understandable English along with working examples as tutorial lessons.

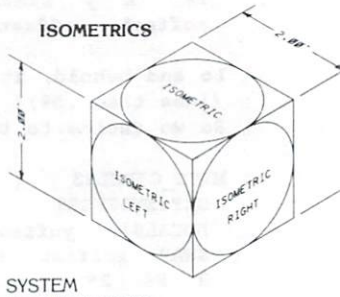


SCHEMATICS



MECHANICAL DRAFTING

ISOMETRICS



SIGNAL NAME	COMP NAME	PIN NUMB	SOURCE	TYPE COMP
S0001	J1	2	*	CON
S0001	U1	1		7400
S0002	J1	3	*	CON
S0002	U1	4		7400
S0003	J1	4	*	CON
S0003	U1	9		7400
S0004	J1	6	*	CON
S0004	U1	12		7400
S0005	J1	9	*	CON
S0005	U1	2		7400
S0005	U1	5		7400
S0005	U1	10		7400
S0005	U1	13		7400
S0006	U1	3	*	7400
S0006	U2	1	*	7400
S0007	U1	6	*	7400
S0007	U2	2	*	7400
S0008	U1	8	*	7400
S0008	U2	4	*	7400
S0009	U1	11	*	7400
S0009	U2	5	*	7400
S0010	U2	3	*	7400
S0010	U2	9	*	7400
S0011	U2	6	*	7400
S0011	U2	10	*	7400
S0012	U2	8	*	7400
S0012	J2	4		CON

NET LIST FROM SCHEMATIC

WHAT DYNAMIC-CAD CAN DO FOR YOU

- D.C gives you all the expected CAD functions of zooming, rotating, panning, group functions and menu driven features.
- D·C brings you professional CAD capability tested and proven in the production of tens of thousands of drawings.
- D·C will liberate you from the need to draw free hand.
- D·C has net listing capability from your schematic.
- Schematic comparison to your printed circuit artwork for continuity check.
- D·C can produce isometric views.
- Mil-Spec quality Leroy* fonts.
- Automatic line dimensioning.
- D·C includes a series of information libraries: Symbols, Electronic Parts/Chips, Architectural Components, Landscaping, etc.
- Data base to store and retrieve information on parts specifications, vendors, and pricing.
- Data base system utilizes ASCII format files which are convertable to other standards.
- Capable of utilizing up to 4,096 colors.
- D·C can generate over 8,000 layers.
- D·C supports most standard dot matrix printers, ink jet, lazer jet, pen plotters, and the Gerber* Photoplotter.

SYSTEM REQUIREMENTS
512 K RAM
2 Disk Drives (or)
1 Drive and Hard Disk
Printer or Plotter

Inquiries invited. (818) 360-3715

*Gerber — trade mark of Gerber Scientific Instruments
*Leroy — trade mark of Keuffel & Esser
*IBM — trade mark of International Business Machines
*AutoCAD — trade mark of Autodesk, Inc.
*Amiga — trade mark of Commodore Business Machines

micro illusions

P.O. BOX 3475, GRANADA HILLS, CA 91344

"I C What I Think!"

By **Ronald Peterson**

Graphics is the single most outstanding feature of the Amiga. When I first saw an Amiga demonstration at the SIGGRAPH '85 convention in San Francisco I was impressed most by its display capabilities. For under $2000, it matched most of the abilities of the $80,000 graphics terminals I use in my job. I knew I had to have one, sooner or later. It turned out to be sooner, and I spent a gleeful couple of days playing with the demo programs, but eventually I wanted to start writing some graphics programs of my own. I started by writing a few routines in BASIC and though it could draw a line or a box with reasonable speed I was horrified by how slow it was in performing the calculations required for more complex pictures. I soon realized that most of those fancy demo programs were written not in BASIC, but in C!

C is a programming language that retains a freedom of access to hardware architecture similar to assembly language while offering many of the more advanced constructs of a high level language, such as loops and if-then-else statements. So I got hold of the Lattice C compiler and the Amiga ROM Kernal Manual and dug in. Learning a new language, a new machine, a new operating system, and a new DOS all at the same time looked like a hopeless task at first, but a couple of months later I was rewarded. I had written a program that drew a single line on the screen! I was ecstatic! While it may not sound like much, it was the first real connection between an idea in my head and a picture on the screen of an Amiga. I had forged myself a new path for expression! Once I had that single line a multitude of doors opened for me. In short order I soon had boxes, triangles, hires screens, and simple animations to fascinate myself with. And so I present here the results of my trials in a short tutorial form to encourage others in their efforts to transform ideas to the screen of an Amiga. First, I will cover some of the basics to ensure a common ground for communication but the real intent of this article is to pass on some of the things I learned that the manuals don't tell you.

There are two ways of accessing the Amiga's graphic capabilities when using the C language. You can either work through the Intuition window manager or you can open your own display window, called a Viewport. Working through Intuition gives you all the flexibility that comes with the windowing system. I am more interested in speed and raw graphics power, so I chose working through my own Viewport.

The fundamental element of a display on the Amiga is a Raster. A Raster is simply an area of memory which has been set aside to be used for holding an image. The Raster memory is periodically scanned by the Amiga custom graphics chips and its contents mapped onto the monitor screen. This reserved area of memory is also called a Bitplane because it is organized into planes (remember geometry?) that are up to 1024 bits square by up to six bits deep. Taking a bit from each plane - for example, the lower right hand corner bit - and putting them together gives a number. This can be used as an index into the Color Lookup Table. The value in this table determines the color for the point on the screen that corresponds to that part of the bitplane - the lower right hand corner.

Since the Amiga doesn't have 1024 by 1024 resolution only a portion of a set of bitplanes can be displayed on the monitor. The selection of this portion is accomplished by using a Viewport. A Viewport is what its name suggests; a rectangular window through which a subset of the bitplanes can be seen.

There can be more than one viewport displayed on the screen at one time but viewports cannot overlap and they cannot exist side by side. They can only be stacked vertically. As far as I can tell, each bitplane can have only one viewport. A "screen" as demonstrated in the Introduction to the Amiga manual is an example of a viewport. A linked set of viewports is appropriately called a View since this is what appears on the monitor screen. In the same sense that a viewport is a window for seeing into a bitplane set, a Rastport is a window for drawing into a bitplane set. A rastport can also be thought of as a communication port because it is the channel through which all the Amiga's built in drawing routines work. So, that should be enough of an explanation to ensure that you know the basic terminology and concepts involved in creating a display. The program listing that accompanies this article demonstrates how to implement all of these concepts in a C program. It draws heavily on the example program on pages 2-28 through 2-31 of the Amiga ROM Kernal Manual which I gratefully acknowledge. But before we get to the program here are some of the tips I promised on things that aren't in the ROM manuals.

The first thing I learned is that there are mistakes in the example programs in the Kernal manual. Although the programs segments and examples are supposedly tested and functional, they don't always work. I suggest reading the first four chapters (particularly the chapters on Messages and Ports and on I/O) to get some idea of how to communicate with the Amiga and in order to be able to troubleshoot the examples.

Double buffering is a technique where two independent viewports are displayed alternately so that all the drawing that is going on is hidden from the observer and a smooth looking animation results. If you are interested in double buffering a display then two rastports are required to draw into two sets of bitplanes. Swapping one rastport back and forth doesn't work.

The LoadView() function is the one that causes what you've drawn to appear on the screen. This is particularly important for double buffering. It is not enough to just change the copper list pointers. You must execute a new LoadView() in order to get the new picture to the screen. If you allocate

memory (such as required for a bitplane) then deallocate when your program is done. If you don't then each time you run your program another chunk of memory will disappear until you reboot, eventually crashing your system when you run out of free memory.

Syncing with the vertical blanking interval is easy once you know how. There are two functions for this purpose called WaitBOVP() and WaitTOF(). WaitBOVP() waits for the electron beam to scan to the bottom of the current viewport and WaitTOF() waits till the vertical blank interval has begun. I found that putting a call to WaitBOVP() right before the LoadView() call syncs things up nicely.

Sometimes you would like to have a program run until you are tired of watching it. So you put it into an endless loop. This works except that you have to reboot the system when you want to use it again. Using control-C to abort the program would be ideal, but writing a console handler to implement such a trap is painful, to say the least. However, someone somewhere has been kind to us and has given us a neat function call just for this purpose. To activate CTRL/C checking set:

Enable_Abort = 1;

Every time your program calls an I/O function (like printf) a check will be made for a CTRL/C. If you would rather handle the abort yourself (so you can deallocate all that memory) set Enable_Abort = 0 and call Chk_Abort() somewhere in your main loop. If ChkAbort() returns a nonzero value then abort! (See listing for example.)

Here are some tricks to speed things up: there are several things you can try. One is to try the Motorola Fast Floating Point math library. It can be significantly faster than the Lattice math routines since they are in hand-tooled machine code, instead of C. [Ed.: You might also consider using fixed-point, floating point numbers, or using only integers, instead of floating point numbers.] Another trick is to reduce the size of the objects you are animating. This is not always desirable. Finally, if you are using a similar set of calculations over and over, such as moving an object in a circle you can precalculate the coordinates of the path and store them in an array to provide much faster access. Now for some fun!

The program shown in Listing 1 shows most of the features I have talked about here. It creates a smoothly animated display on a 320 by 200 viewport of some rotating triangles. The positions of the corners are precalulated to improve the speed and a control-C trap is implemented as described above. It is synced with the vertical blanking interval and is double buffered so that you don't see the triangles being drawn.

When you run this program, don't be scared when your screen instantly turns togarbage. This is just what the bit plane looks like before you clear it. I left this in intentionally. I like it. It is something you don't get to see. Use SetRast() if you want to blank the screen. After a few seconds, the precalculations will be done, and the triangles will appear. The source code to this program can be found on the AMICUS disks. Check the catalog for more information.

Listing One

```
/* ----- F A S T E S T . C ------------------- */
/*  Demonstrate opening a Viewport and doing
simple */
/*  Animation using a double buffered display. */
/*  Written By Ronald Peterson                 */
/*  With acknowledgements to the ROM Manual    */
/*-------------------------------------------*/

/* INCLUDE's -
-------------------------------------- */

#include "stdio.h"
#include "math.h"
#include "exec/types.h"
#include "graphics/gfx.h"
#include "hardware/dmabits.h"
#include "hardware/custom.h"
#include "graphics/gfxmacros.h"
#include "graphics/copper.h"
#include "graphics/view.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "exec/exec.h"
#include "graphics/text.h"
#include "graphics/gfxbase.h"
#include "graphics/rastport.h"

/* DEFINE's -
-----------------------------------------*/

#define DEPTH 4
#define WIDTH 320
#define HEIGHT 200
#define NOT_ENOUGH_MEMORY -1000
#define CLIP 0  /* 0 = no clip // 1 = clip */
/* set CLIP to 1 for debugging so that you don't
   draw outside the bitplane */
/* and crash the system. */

/* GLOBAL variable definitions -
-----------------------------------------*/

struct View v;
struct ViewPort vp;
struct ColorMap *cm;
struct RasInfo ri;
struct BitMap b;
struct BitMap b2;
    /* second bitmap for double buffering */
struct cprlist *lof1;
     /* copper list pointers for double buffer */
struct cprlist *lof2;
struct cprlist *shf1;
struct cprlist *shf2;
extern int Enable_Abort;

/* RastPort variables ---------------- */

PLANEPTR rastpoint, rastpoint2;

struct RastPort rp;
struct RastPort rp2;

WORD areabuffer[250];
struct TmpRas tmpras;
struct AreaInfo myAreaInfo;
```

```
WORD areabuffer2[250];
struct TmpRas tmpras2;
struct AreaInfo myAreaInfo2;

/* ---------------------------------- */

LONG i;
SHORT j,k,n;
int c, iphi;
double phi, rad;
double Qsinray(), Qcosray();

extern struct ColorMap *GetColorMap();
struct GfxBase *GfxBase;
struct View *oldview;
/* save pointer to old view so can restore */

USHORT colortable[] = {0x000, 0xf00, 0x0f0, 0x00f,
0x880, 0x088, 0x808, 0x123, 0x045, 0xe4f, 0x59a,
0x3d3, 0x7fa, 0x675, 0xabc, 0xfff};
/* set colors black, red, green, blue,
   yellow,purple,orange,etc */

UBYTE *displaymem;
UWORD *colorpalette;

/* Start of main routine -
------------------------------*/

main()
{
   static int xxx[12], yyy[12], flag, good, iinc,
ioffset;
   static double inc, two_pi, offset;

   rad = 3.1415926/180.;
   inc = 5. * rad;
   offset = 80. * rad;
   iinc = 2;
   ioffset = 80;
   two_pi = 6.2831852;
   GfxBase = (struct GfxBase *)
          OpenLibrary("graphics.library",0);
   if (GfxBase == NULL)exit(1);
   oldview = GfxBase->ActiView;
         /* save current view to restore later */

   InitView(&v);       /* initialize view */
   InitVPort(&vp);     /* initialize viewport */
   v.ViewPort = &vp; /* link view into viewport */

   /* init bit map (for rasinfo and rastport) */
   InitBitMap(&b,DEPTH,WIDTH,HEIGHT);
   InitBitMap(&b2,DEPTH,WIDTH,HEIGHT);

   /* set up information about Raster location */
   ri.BitMap = &b;
   ri.RxOffset = 0;
   ri.RyOffset = 0;
   ri.Next = NULL;

   /* specify dimensions */
   vp.DWidth = WIDTH;
   vp.DHeight = HEIGHT;
   vp.RasInfo = &ri;

   /* init color table */
   cm = GetColorMap(16);
         /* 16 entries, since 4 planes deep */
   colorpalette = (UWORD *)cm->ColorTable;
   for(i=0; i<16; i++)
      *colorpalette++ = colortable[i];

   /* copy my colors into this data structure */
   vp.ColorMap = cm;
         /* link it with the viewport */

   /* allocate space for two bitmaps */
   for(i=0; i<DEPTH; i++)
   {
      b.Planes[i] =
(PLANEPTR)AllocRaster(WIDTH,HEIGHT);
      if (b.Planes[i] ==
NULL)exit(NOT_ENOUGH_MEMORY);
      b2.Planes[i] =
(PLANEPTR)AllocRaster(WIDTH,HEIGHT);
      if (b2.Planes[i] ==
NULL)exit(NOT_ENOUGH_MEMORY);
   }

   ri.BitMap = &b;
   MakeVPort(&v,&vp);

   /* construct copper instr (prelim) list */
   MrgCop(&v);
   /* merge prelim lists together into a real
    * copper list in the view structure   */
   lof1 = v.LOFCprList;
        /* store pointers to copper lists */
   shf1 = v.SHFCprList;
        /* for double buffering  */
   v.LOFCprList = 0;
   v.SHFCprList = 0;
```

```
   ri.BitMap = &b2;
   MakeVPort(&v,&vp);
   MrgCop(&v);
   lof2 = v.LOFCprList;
      /* store pointers to copper lists*/
   shf2 = v.SHFCprList;
      /* for double buffering  */

   LoadView(&v);
      /* tell copper to use list for display */

   /* now initialize a rastport */
   InitRastPort(&rp);
   rp.BitMap = &b;   /* link rastport to bitmap */
   SetDrMd(&rp,JAM1);  /* set draw mode to JAM1 */

   /* now set up buffer for area fill */

   InitArea(&myAreaInfo, areabuffer, 100);
   rp.AreaInfo = &myAreaInfo;

   rastpoint =
(PLANEPTR)AllocRaster(WIDTH,HEIGHT);
   InitTmpRas(&tmpras, rastpoint,
RASSIZE(WIDTH,HEIGHT));
   rp.TmpRas = &tmpras;

   /* now initialize a second rastport */
   InitRastPort(&rp2);
   rp2.BitMap = &b2; /* link rastport to bitmap */
   SetDrMd(&rp2,JAM1); /* set draw mode to JAM1 */

/* set up buffer for area fill in RastPort 2 */

   InitArea(&myAreaInfo2, areabuffer2, 100);
   rp2.AreaInfo = &myAreaInfo2;

   rastpoint2 =
(PLANEPTR)AllocRaster(WIDTH,HEIGHT);
   InitTmpRas(&tmpras2, rastpoint2,
RASSIZE(WIDTH,HEIGHT));
   rp2.TmpRas = &tmpras2;

   phi =0.;
   iphi =0;
   flag = 0;
   SetOPen(&rp,1);     /* set outline color */
   SetOPen(&rp2,1);     /* set outline color */
   Enable_Abort = 0;
        /* disable automatic CTRL-C abort */
   for(i=0; i<100000; i++)
   {
      if (0 != Chk_Abort())
          /* check for CTRL-C abort */
      {
         goto dats_all;
      }
      v.LOFCprList = lof1;
          /* set pointers to copper lists and */
     v.SHFCprList = shf1;
          /* set RastPort bitmap pointer*/
      WaitBOVP(&vp);
          /* wait till in vertical blank area */
      LoadView(&v);
          /* display Viewport 1 (bitmap 1) */

      SetRast(&rp2,0);
          /* clear Viewport 2 (bitmap 2) */
      if (flag == 0)
      {
         iphi = iphi + iinc;
         for (good=0; good<9; good++)
         {
            iphi = iphi + ioffset;
            if (iphi > 360)iphi=iphi-360;
            xxx[good] = (int)Qsinray(iphi);
            yyy[good] = (int)Qcosray(iphi);
#if CLIP
   if (xxx[good] > (WIDTH-1)) xxx[good] = WIDTH-2;
            if (xxx[good] < 0) xxx[good] = 1;
 if (yyy[good] > (HEIGHT-1)) yyy[good] = HEIGHT-2;
            if (yyy[good] < 0) yyy[good] = 1;
#endif
         }
       }
        /* move endpoints slightly and redraw */
       flag = 1;
       iphi = iphi + iinc;
       for (good=0; good<9; good++)
       {
          iphi = iphi + ioffset;
          if (iphi > 360)iphi=iphi-360;
          xxx[good] = (int)Qsinray(iphi);
          yyy[good] = (int)Qcosray(iphi);
#if CLIP
   if (xxx[good] > (WIDTH-1)) xxx[good] = WIDTH-2;
          if (xxx[good] < 0) xxx[good] = 1;
 if (yyy[good] > (HEIGHT-1)) yyy[good] = HEIGHT-2;
          if (yyy[good] < 0) yyy[good] = 1;
#endif
       }
       SetAPen(&rp2,3);
       AreaMove(&rp2,xxx[0],yyy[0]);
       AreaDraw(&rp2,xxx[1],yyy[1]);
       AreaDraw(&rp2,xxx[2],yyy[2]);
       AreaEnd(&rp2);

       SetAPen(&rp2,2);
       AreaMove(&rp2,xxx[3],yyy[3]);
       AreaDraw(&rp2,xxx[4],yyy[4]);
       AreaDraw(&rp2,xxx[5],yyy[5]);
       AreaEnd(&rp2);

       SetAPen(&rp2,1);
       AreaMove(&rp2,xxx[6],yyy[6]);
       AreaDraw(&rp2,xxx[7],yyy[7]);
       AreaDraw(&rp2,xxx[8],yyy[8]);
       AreaEnd(&rp2);

       v.LOFCprList = lof2;
            /* set pointers to copper lists and */
      v.SHFCprList = shf2;
            /* set RastPort bitmap pointer* /
       WaitBOVP(&vp);
            /* wait till in vertical blank area */
       LoadView(&v);
            /* display Viewport 2 (bitmap 2) */

       SetRast(&rp,0);
            /* clear Viewport 1 (bitmap 1) */
       iphi = iphi + iinc;
       for (good=0; good<9; good++)
       {
          iphi = iphi + ioffset;
          if (iphi > 360)iphi=iphi-360;
          xxx[good] = (int)Qsinray(iphi);
          yyy[good] = (int)Qcosray(iphi);
#if CLIP
  if (xxx[good] > (WIDTH-1)) xxx[good] = WIDTH-2;
```

```
        if (xxx[good] < 0) xxx[good] = 1;
 if (yyy[good] > (HEIGHT-1)) yyy[good] = HEIGHT-2;
        if (yyy[good] < 0) yyy[good] = 1;
#endif
      }
      SetAPen(&rp,3);
      AreaMove(&rp,xxx[0],yyy[0]);
      AreaDraw(&rp,xxx[1],yyy[1]);
      AreaDraw(&rp,xxx[2],yyy[2]);
      AreaEnd(&rp);

      SetAPen(&rp,2);
      AreaMove(&rp,xxx[3],yyy[3]);
      AreaDraw(&rp,xxx[4],yyy[4]);
      AreaDraw(&rp,xxx[5],yyy[5]);
      AreaEnd(&rp);

      SetAPen(&rp,1);
      AreaMove(&rp,xxx[6],yyy[6]);
      AreaDraw(&rp,xxx[7],yyy[7]);
      AreaDraw(&rp,xxx[8],yyy[8]);
      AreaEnd(&rp);
   }

dats_all:
   LoadView(oldview);  /* put back the old view */

   FreeMemory();              /* exit gracefully */
   CloseLibrary(GfxBase);
}

/* End of main ------------------------*/

/* return user and system-allocated memory
   to sys manager */
FreeMemory()
{
   /* free drawing area */
   FreeRaster(rastpoint,WIDTH,HEIGHT);
   FreeRaster(rastpoint2,WIDTH,HEIGHT);
   for(i=0; i<DEPTH; i++)
   {
      FreeRaster(b.Planes[i],WIDTH,HEIGHT);
      FreeRaster(b2.Planes[i],WIDTH,HEIGHT);
   }
 /* free the color map created by GetColorMap() */
   FreeColorMap(cm);
   /* free dynamically created structures */
   v.LOFCprList = lof1;
   v.SHFCprList = shf1;
   FreeVPortCopLists(&vp);
   FreeCprList(v.LOFCprList);
   FreeCprList(v.SHFCprList);
      /* free up interlace copper list */
   v.LOFCprList = lof2;
   v.SHFCprList = shf2;
   FreeVPortCopLists(&vp);
   FreeCprList(v.LOFCprList);
   FreeCprList(v.SHFCprList);
       /* free up interlace copper list */
   return(0);
}

/* store endpoints of triangles to speed up
   rotation */

double Qsinray(theta)
int theta;
{
   int j;
   static int iflag = 0;
   static double x, result[365];

   if (iflag == 0)
   {
      iflag = 1;
      x = 0.;
      for(j=0; j<365; j++)
      {
         x = x + .0174532;
         result[j] = 150.*sin(x)+151.;
      }
   }
   return(result[theta]);
}

double Qcosray(theta)
int theta;
{
   int j;
   static int jflag = 0;
   static double x, result[365];

   if (jflag == 0)
   {
      jflag = 1;
      x = 0.;
      for(j=0; j<365; j++)
      {
         x = x + .0174532;
         result[j] = 95.*cos(x)+99.;
      }
   }
   return(result[theta]);

}
```

•AC•

SYMPHONY MUSIC LIBRARY

Over 800 Songs

Over 30 Hours of Music

The library contains over 800 four voice stereo music pieces to turn your AMIGA into a sophisticated Jukebox providing you with over 30 hours of music.

- Turn your AMIGA into an incredible music machine.
- Over 800 masterpieces in entire library.
- Over 100 songs in each volume.
- Not just single note music, but 4 full voice arrangements.
- Each volume contains over 3 hours of music.
- Stereo, Mono, and MIDI output simultaneously.
- You may specify sound of each of the voices.
- Over 30 hours of music in entire library.
- User selectable tempo.
- Selections may be played thru any MIDI synthesizer using the MIDI CONNECTION (sold separately).
- User selectable key (transposition).
- Includes Jukebox program to allow you to:
  1) Select all pieces and sit back to hours of uninterrupted music.
  2) Select the order you wish the pieces to be played.
  3) Select the number of times you wish each piece to be played.

SYMPHONY LIBRARY Vol 1
Music from Stage, Screen & TV
Pop Music from the 50's, 60's, & 70's
Old Time Favorites
Classical
Christmas (popular & traditional)
Patriotic
Polka Party

SYMPHONY LIBRARY Vol 2
Music Potpourri
Barbershop Quartette
Religious Music
Hits of the 80's
Variety

SYMPHONY LIBRARY Vol 3
Pop Music from the 40's, 50's, 60's, 70's
Beatles Medley

SYMPHONY LIBRARY Vol 4
TV Theme Music
Beethoven, Broadway, & Blues
Kenny Rodgers Greatest Hits
Best of the Beatles
Country Classics

SYMPHONY LIBRARY Vol 5
Popular Potpourri
Polka Party
Classical
Tribute to Bach
Christmas Treasure

SYMPHONY LIBRARY Vol 6
Nostalgic
Richard Rogers Songbook
Music From the Movies
Pop Music from the 60's, 70's, & 80's

SYMPHONY LIBRARY Vol 7
Variety
Classical
Pop Music from the 30's thru 80's
Popular Classics

SYMPHONY LIBRARY Vol 8
Surprise!

Each Volume
~~$39.95~~
$29.95
Introductory Offer

THE MIDI CONNECTION

Attention MIDI Users!

The MIDI CONNECTION is a hardware interface to allow you to connect the AMIGA to any MIDI synthesizer, drum machine, sampler and more. MIDI IN and OUT connectors with cables allow easy installation. The MIDI CONNECTION may be used with our SYMPHONY MUSIC LIBRARY or the MIDI SYMPHONY.

~~$49.95~~
$39.95
Introductory Offer

THE MIDI SYMPHONY

The MIDI SYMPHONY program and the MIDI CONNECTION hardware interface give you the features you need to compose serious music using MIDI equipment. Features include:

- Supports up to 16 tracks.
- 10,000 events per track.
- 40,000 events all tracks.
- May be used as a sequencer.
- Menu driven.
- Overdubbing.
- User friendly graphics display.
- Metronome available for synchronization.
- Tempo may be modified.
- Change key (Transposition).
- Quantizing to 32nd or 64th.
- Playback any or all tracks at any tempo.
- Tracks may be deleted, copied, transposed or mixed.
- Filter out unwanted channel or type of MIDI data.
- Simple music editing.
- Real Time recording and playback.
- Playback all or individual tracks.
- Track delete, copy, transpose or mix.
- Editing

~~$99.95~~
$89.95
Introductory Offer

COMING SOON

SYMPHONY SOUND SAMPLER. This high speech digital sampler allows you to create and modify sounds. When used with the SYMPHONY Piano Keyboard, you can turn your AMIGA into an Ensoniq Mirage.

SYMPHONY WRITER. A composition program specifically for developing sheet music.

SYMPHONY 61 note Piano Keyboard. This professional quality keyboard transforms your AMIGA into a real synthesizer.

CASIO CONNECTION. This program and the 61 note keyboard allows CZ-101 and CZ-1000 owners to use a full size keyboard.

AMIGA is a trademark of Commodore-Amiga Inc.
Mirage is a trademark of Ensoniq Inc.

We accept CASH, CHECK, COD, VISA and MASTER CARD orders.
Shipping and handling US and Canada $3.00
Shipping and handling outside the US and Canada $5.00
COD charge $2.00
Illinois residents add 6¼% sales tax.

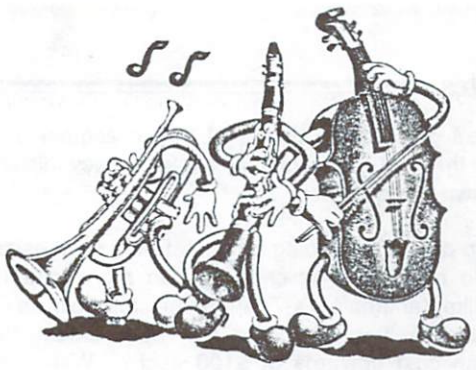
VISA MasterCard

Dealer Inquiries Invited

Speech Systems

38W255 DEERPATH ROAD
BATAVIA, ILLINOIS 60510
(312) 879-6880

AmigaNotes



By **Richard Rae**
Music Editor
Compuserve [72177,3516]

AN INTRODUCTION TO MIDI

If you've been tracking the various Amiga music packages soon to hit the market, you've probably run across references to **MIDI** capability. You've also heard about MIDI if you've been involved in the ongoing Amiga/ST debate, or if you own one of the current crop of synthesizers. I want to devote this column to a very brief introduction to MIDI and why it is a valuable asset.

MIDI stands for **Musical Instrument Digital Interface.** Like RS232 or Bell 212A, it is a standard for communications. And like other standards it can be extremely useful... consider where we'd be if every computer manufacturer used his own (incompatible) serial communications protocol!

MIDI is the result of a collaboration between numerous musical instrument manufacturers, and is guided by the MIDI Manufacturers Association (MMA) and the International MIDI Association (IMA). The IMA sells both abbreviated and detailed MIDI spec packages, and membership is available to anyone willing to pay the dues. (The address will be given at the end of this column in case you're interested...)

WHAT'S IT GOOD FOR?

Now before delving into the specifics, let's ask a very important question: what can MIDI do for you? The answers are many and varied, depending on your interests in music. Let's take a look at just a few of them.

Possibly the most important application to many of us is the "tapeless studio". If you wanted to record a difficult synthesized passage in the days before MIDI, you had to play it over and over until you got it right. With MIDI, the choices multiply. Oh, you can still try it until you get it right. But instead of using a mechanical deck time and again, waiting through the rewind as well as adding wear and tear, you can record the passage digitally with your computer via MIDI. "Rewind" is instantaneous, and there are no mechanical parts involved. What if you almost got it but made one tiny little mistake? With most MIDI sequencers you can actually edit the recording on a NOTE BY NOTE BASIS. This is almost impossible on a tape deck, except for those of us with infinite patience and a steady hand with a razor blade. Or suppose the passage is just too fast for your fingers. You can slow down the data rate: reduce the tempo of your recording and play the piece at a speed you can handle. After recording at the reduced speed, you can return the tempo to normal with no loss of quality. If you just can't play at all, you still have an option: enter the score note by note in musical notation format. Once the score is entered, you can ask the computer to play it back through the synthesizer just as if you had originally played it on the keyboard.

MIDI will also help you "clean up your act". If you've recorded a passage which is perfect except for timing, most sequencer programs will allow you to **quantize** the data, "snapping" the notes to the nearest beat.

MIDI is a great time saver as well. If you want an arpeggio to run throughout a piece of music, you can play it once, then use a "cut and paste" function to replicate it throughout. If you decide to rearrange a composition after
getting it into the sequencer, you can move or transpose entire sections as needed. (These functions are actually a part of the music composition program being used, but without MIDI you would be limited to either hearing the music via the program's internal voices or playing the piece manually on your synthesizer until you got it right.)

With a computer acting as a MIDI sequencer, you also have the advantage of **multi-tracking.** With two synthesizers you can record a bass part on one track, then listen to the bass track while recording a rhythm track. Each track which you "lay down" in this manner is totally independent of the others. Using this layering technique you can literally become a one man band, limited only by the number of synthesizers you own.

At this point I expect at least a few of you are protesting. "I just bought the computer, now you're telling me to buy a bunch of synthesizers?" No, not at all. But remember, MIDI is a system for COMMUNICATIONS. An RS232 port is useless unless you have a modem or serial printer or SOMETHING to talk to; so it is with MIDI. If you're not interested in controlling electronic music instruments, if the internal sound capabilities of the Amiga are sufficient for you, then you will not be interested in MIDI.

If you dream of setting up your own electronic music studio however, look at the cost advantages of MIDI. Without it, you can produce eight part music with only one synthesizer and an eight track recorder, with complete control over the final mixdown. But even the least expensive eight track format -- quarter inch -- will cost you at least $1500 discounted for the deck alone. Then you need a mixer and at least one synthesizer, and...

On the flip side, four track cassette decks are currently available for about $600 retail, which means you'll probably be able to buy them discounted for about $300 very soon. MIDI synthesizers are getting cheaper all the time; I have seen the Casio CZ101 for $250 discounted, and the Yamaha DX-100 is readily available for less than $400. With a four track deck, MIDI/tape sync interface, sequencer program, and two synthesizers, you can create the same "eight track" final

product. We'll get into the details of this in another article; suffice to say that you can save quite a bit of money this way if you already own the computer.

MIDI can also do housekeeping for you. Every synthesizer is limited in the number of voices you can store in it (with instruments like Yamaha's DX-7, where the storage is on RAM cartridges, the limiting factor is the pocketbook: those cartridges can cost upwards of $100 each). With a MIDI **librarian** program, you can store your sound patches on disk and load them into the synthesizer a voice or bank at a time. Programming synthesizers through the tiny LCD or LED "window" provided is difficult at times. A **patch editor** allows you to use your computer to program the instrument, viewing an entire screen of information at one time.

For live acts, your computer can coordinate the entire show. MIDI doesn't stop at keyboard instruments... there are MIDI controlled mixers, effects boxes, amplifiers, lighting controllers, drum machines, and more. MIDI can bring them all together!

The key to this versatility is the fact that MIDI handles DATA, not sound: rather than recording musical tones, a MIDI system stores information about what note is played on which instrument when and for how long.

Now to delve a little deeper. Like RS232, MIDI is both a hardware and a software standard. Let's take a look at the hardware aspects of MIDI.

MIDI HARDWARE

A MIDI connection is made with a multi-wire shielded cable less than 50 feet in length and terminated with five-pin **DIN** connectors. DIN connectors are those strange multi-pin connectors one sometimes finds on the back of expensive receivers and open reel decks; similar connectors are found on the Commodore 64 and Tandy Model 100.

DIN connectors have been accepted in Europe for years, and were chosen for MIDI because they provided a relatively inexpensive yet reliable multi-pin connection. (For manufacturers desiring even higher quality, the MIDI spec allows substitution of XLR connectors if the manufacturer also supplies appropriate adapters with the equipment.)

The five pins of a MIDI DIN connector are arranged in a semi-circle and are surrounded by a metal shell with an indexing notch. The center pin is a ground and is connected to the cable shield, while the two outermost pins are not used; this leaves two pins to carry signals. A logical person would assume that, like RS232, these two pins carry transmit and receive data. Our logical person would be wrong! MIDI technology reaches back to the days of the teletype: it is in fact a five milliamp **current loop** which requires two wires to transmit data.

Now why on Earth would designers of a modern high-technology interface choose such an antiquated technique? The answer is three-fold and relates to one very simple fact: MIDI was designed as an inexpensive interface for musical instruments.

The first reason for a current loop interface is noise rejection. Anyone who has performed in an "electric" band knows how various buzzes, clicks, and hums can work their way into an audio system. The same random signals which simply cause irritating noises on the audio lines would play havoc with a digital control system like MIDI. A current loop is a low impedance system, and as such is very resistant to external interference.

The second reason for choosing a current loop is closely related to the first: isolation. Performers in that same live band will tell you that singing through an amplifier with a different ground polarity from the guitar amplifier they are using can be a literally shocking experience. In addition to the potential damage to sensitive digital equipment, ground loops of this sort are responsible for many of the hum problems encountered during an audio engineer's career. A properly designed current loop like that used by MIDI can allow complete isolation between various instruments.

The third reason for the MIDI current loop is one of cost. Rather than requiring line drivers or level shifters, the MIDI current loop can be driven by a simple transistor or TTL gate. Isolation on the receiving end is provided by the only potentially high cost component: an optoisolator. In fact, a complete MIDI interface for a UART port can be built with only six components: a TTL or transistor driver stage, an optoisolator input stage, three current limiting resistors, and a protection diode! It would be difficult to develop a more straightforward interface providing all the benefits of the MIDI current loop.

MIDI data is transmitted serially at a rate of 31.25 KBaud, or 31,250 bits per second. Those of you who know baud rates will recognize this as a very non-standard speed. With the exception of very old equipment, all standard baud rates are multiples of 300 baud: 300, 600, 1200, 1800, 2400, 4800, 9600, and 19200 are most often seen. Why an oddball baud rate? As with the current loop, a close examination reveals sound reasoning.

For example, why should MIDI use a "standard" baud rate? MIDI has nothing to do with RS232 communications. In fact, using one of these data rates almost forces us to use a dedicated "baud rate generator"... an unneeded expense. 31.25 Kbaud was chosen primarily because it is easily derived from most computer system clocks. In other words, divide one MHz by 32 and you get 31.25 Khz! For a two MHz clock one would divide by 64, and so on. Dividing by a power of two is trivially easy, and in some cases these signals are already available elsewhere in the system. 31.25 Kbaud was also chosen because it is the highest "easily derived" data rate less than 50 Kbaud. The latter figure is the practical upper limit for some of the components which might be used in the design of a MIDI port.

As mentioned earlier, MIDI data is transmitted serially. The format is a ten bit word with one start bit, 8 data bits, and one stop bit. Unlike the baud rate, this is a very standard format. A MIDI port can therefore be driven by a plain vanilla serial device such as a UART or ACIA.

IS THE ATARI ST ONE UP ON THE AMIGA?

Before we delve into the software aspect of MIDI, let's take a moment to discuss Amiga vs. Atari vs. everybody else.

The Atari 520ST and 1040ST both have built-in MIDI ports, and Atari fans like to point that out when engaged in a "My computer is best" battle. True, the Amiga does not have a built-in MIDI port, but let's examine why this is not as serious a problem as it could be.

The main difficulty for computers without MIDI ports is the non-standard baud rate. In most cases, the standard serial ports provided are simply not capable of communicating at 31.25 Kbaud. With the Amiga, this is not the case. The designers knew that MIDI was an important issue, and designed the serial port to support the required baud rate. In fact, pre-release versions of the Preferences program included a serial port baud rate designated as "MIDI": 31.25 Kbaud. The current version of Preferences does not include this selection, but the Amiga still supports it.

(I am told that the Preferences to be shipped with AmigaDOS version 1.2 once again includes the 31,250 baud setting.) (The version 1.2 Preferences does restore the MIDI baud rate setting. In fact, it has another screen in Preferences to allow other serial settings as well, such as 7 or 8 bits, parity, stop bits, etc., beyond the simple baud rate setting in the 1.0 and 1.1 Preferences.)

With the baud rate hurdle behind us, the only requirement is a simple interface which converts RS232 signals to and from MIDI signals. This interface is almost trivially simple, and can be built with a handful of components. There are easily a half dozen manufacturers developing MIDI interfaces which simply plug into the serial port; most will retail for around $50.

MIDI SOFTWARE

The software portion of the MIDI spec is as straightforward and well thought out as the hardware. MIDI communication is based on the **message**. All MIDI message bytes fall into one of two categories: **status** or **data**. A status byte always has its **most significant bit** (MSB) set; a data byte always has a zero MSB.

(Immediately we have gained an advantage: simplicity of programming. Rather than keeping data counters and worrying about resyncing in the case of lost data, our status bytes are always distinguishable from data. And we can even transmit messages within messages.)

A message is typically a status byte followed by one or more data bytes. There are several types of messages: **channel system, common system, real-time**, and **system exclusive**.

CHANNEL MESSAGES

Channel messages are generalized pieces of information which any synthesizer might need in order to make a sound, such as key depressions and patch selection.

The status byte of a channel message includes a four bit number in the lower nibble which is the message's channel. MIDI devices monitoring the data stream can be set to respond to all messages or only the messages for a single channel. Thus we can connect sixteen devices via one daisy-chained cable and each will respond independently.

With the MSB set to one indicating a status byte and four bits for the channel number, three bits are left to encode seven different channel messages. These messages are **note off, note on, polyphonic aftertouch, control change program, change channel, after touch**, and **pitch wheel change**. (Some of these messages are self-explanatory while others are probably not familiar to you. Since this is just an overview of MIDI, I'll save the details for a later article.)

Note that with three bits there are eight, not seven, possible bit combinations. The eighth combination is used to signal a **system** message. System messages do not include channel numbers; the lower nibble is instead used to indicate the type of system message.

SYSTEM COMMON MESSAGES

System common messages are intended for all devices in a MIDI system, and encompass general purpose data. These messages include **song position pointer definition, song selection**, and **tune requests**.

SYSTEM REAL-TIME MESSAGES

System real-time messages are also intended for all devices in a MIDI system, and deal with timing of events. These messages are composed solely of status bytes. For this reason they can be interleaved with other messages, even appearing within another message's data stream. Real-time messages are given this priority because, after all, timing is everything. The real-time message group includes the **timing clock**, **song start**, **song continue**, and **song stop**.

SYSTEM EXCLUSIVE MESSAGES

These messages are intended for specific devices. Unlike channel messages, reception of system exclusive messages is based on the device itself rather than channel number. For example, a system exclusive message may be sent to all Yamaha devices in a system, and it will be ignored by Casio or Korg machines. Within a system exclusive message all bets are off. Such a message might be a bulk digitized audio dump, a patch parameter transfer, or any function unique to one manufacturer's system. Interpretation of the data between the status byte and EOX (end-of-system-exclusive-message) byte is the sole responsibility of the receiving device.

RECAP

As you can see, the MIDI definition is well structured yet flexible. Hardware is designed to be cheap and rugged. Status bytes are immediately discernible from data. Three groups of messages encompass song parameter setup, system timing, and actual song execution, while a fourth message group provides a manufacturer the ability to implement anything not already included in the spec.

The MIDI spec states that its purpose is to "enable... synthesizers, sequencers, home computers, rhythm machines, etc. to be interconnected through a standard interface", and all in all it does its job well. There are problems, but they are for the most part minor and avoidable.

MIDI continues to become a unifying force in the fields of computers and electronic music. Expect to see further columns on the wide world of MIDI and its applications.

Until next month...

Nybbles,
Rick

For more information about MIDI, contact:

International MIDI Association
11857 Hartsook Street
North Hollywood, CA 91607
(818) 505-8964

•AC•

Your Menu, Sir!

by Bryan D. Catley

When you sit down in a restaurant, the menu you receive is invariably a direct reflection of the quality of the restaurant itself. So too is the case with menu-driven programs; they are a direct reflection of the overall quality of those programs which employ them.

As you are probably aware, Amiga Basic gives you full access to the pull-down menus available to the Amiga, and allows you to establish your own menus, which you control directly from your program. And that is exactly what this tutorial and the accompanying program will show you how to do! You will learn to:

-establish your own menus
-set up a menu interrupt routine, and turn it on
-enable and disable items on the menu
-insert a checkmark next to the selected item
-enable a menu item only after certain other events have occured
-turn off the menu interrupt routine, and restore the1 standard Amiga Basic menus

Further, an expansion of the accompanying program will be suggested which will provide an ideal first "hands-on" experience with custom menus.

Please read this article completely, and examine the program listing before you actually enter it; and when you do, be sure and save a copy before you execute it for the first time.

The Menu Demonstration Program

This program is completely menu driven. When you use it, you will be presented with a header screen, and nothing else will happen until you select a menu item.

Pressing the right mouse button will reveal a new set of three menu titles: "Boxes", "Polygons", and "Run". Moving the pointer on top of any one of them (while keeping the right button pressed) will reveal the associated "pull-down" menu list. Move the pointer up and down on the list, and note how the enabled items are highlighted. When the desired item is highlighted, release the right button to select it.

Boxes. This menu allows you to open a window, and fill it with continually changing squares or oblongs in randomly selected colors. There are also items to clear and close the window.

Additionally, you may also move the Window around the screen by using the standard "Drag Bar".

Polygons. This menu provides similar features to the *Boxes* menu, except the shapes are triangles, or are randomly created with up to eight sides.

Run. This menu has a single item: Quit. Selecting this terminates the program and returns you to Basic. Note that "Quit" is only enabled when both the "Boxes" and "Polygons" windows are closed.

Overview of Custom Menu Handling

Setting up and using your own custom menus is really quite simple, and entails the following steps:

-set up your custom menus
-turn them on and advise Basic of the name of the routine to receive control when a menu item is selected
-continue processing, or wait for a menu item to be selected
-when a menu item is selected, the interrupt routine determines the item selected, sets appropriate indicators, and returns to the main program
-the main program examines the indicators and takes appropriate action
-when no longer required, the custom menus are turned off, and the standard Amiga Basic menus are restored

That's all there is to it! Now let's take a closer look at how we can accomplish all this.

Establishing Your Custom Menus

Look at the first (initialization) part of the program; you will see a number of MENU statements. It is these statements which define the menus that are going to be used by the program. (This statement is also used to modify the status of existing items). The format is:

MENU number,item,mode[,"title"]

number is the number of the menu being referenced. Numbers start at one and go from left to right across the screen. In the standard menu set, "Project" is one, and "Output" is four

item is the number of the item within the menu. They start at zero (for the menu title box), and go up by one for each item in the menu. In the standard "Project" menu, "New" is one, and "Quit" is five

mode defines the desired status of the menu item; it may be 0, 1, or 2.

0 Item is disabled; i.e. it may not be selected. If zero is selected for item zero, (the menu title), the entire menu is disabled regardless of the setting of individual items within that menu

1 Item is enabled; i.e. it may be selected

2 As for 1, except a checkmark is placed in the item box when it is selected

"title" is the text that is to appear in the menu box. If you plan on using mode 2, you must leave two spaces to the left of the text. This is omitted when modifying the status of an existing menu item

Looking at our MENU statements, you will see that all our custom menus are fully defined, but only the "Quit" and "Open" items are enabled. This is because, initially, they are the only valid selections. Also, note that we have defined three menus that will replace the first three standard menus. However, the fourth standard menu (Output) would still be displayed, so we turn it off with the final MENU statement that contains a null title.

Finally, we use the "MENU ON" statement to tell Basic to start using the custom menus, and the "ON MENU GOSUB ...." statement is used to identify the subroutine to be given control when a menu item is selected. Both statements must be issued before the custom menus may be used, but they do not, necessarily, have to be adjacent.

The Menu Interrupt Subroutine

Once menu trapping has been turned on, this is the subroutine that will automatically receive control when any menu item is selected. And it is a genuine subroutine that ends with a RETURN statement! When it terminates, control is always returned to the statement following the one being executed when the menu item was selected. It is just as if you had inserted the following piece of code between each statement in the program: "IF MENU(0)<>0 THEN GOSUB RtnName". Note that Basic temporarily turns off menu trapping while in the interrupt routine, so recursive menu traps are impossible.

When first entered, we use the functions MENU(0) and MENU(1) to determine the selected menu and menu item respectively. (MENU(0) is reset to zero each time it executes so you may poll it, if desired, much as you might poll INKEY$). Now that we have these two values, it becomes very easy to direct the program flow by using the "ON ... GOTO ..." statement; once for the menu, and once for the item.

If an "Open" item was selected, the first thing that is done is to disable the "Open" item (no longer a valid request), and enable all the other items (they are now perfectly valid requests). Further, the variable "opncount" (previously initialized to zero) is incremented by one. If it is then equal to one, the "Quit" menu item under4 5"Run" is disabled. This prevents the user from terminating the program while any windows are still open.

Selecting a "Close" item essentially reverses the "Open" processing. The "Open" item is enabled while the others are disabled. "opncount" is decremented by one, and the "Quit" item is enabled when it becomes zero. The user is now able to select "Open" again whenever it becomes desirable.

When a processing item ("Squares", "3 Sided", etc) is chosen, the selected item is enabled with a checkmark, while its partner is enabled with no checkmark; (remember, it is still a valid selection). Additionally, the window containing the selected shapes is made the current window, (thereby forcing it to the front).

Should "Clear" be selected, the window is made current and cleared, and both processing options are enabled with no checkmarks.

The most important things of all to note are that each processing option ends up by setting the variable "type" to an appropriate value, and that the variable "sw" is set to one just before the RETURN is executed. The contents of these two variables allow the main body of the program to detect that a menu item has been selected, and carry on as appropriate for the item selected.

"type" essentially tells the main program to wait for another menu selection (via the SLEEP statement) or to do some specific processing; draw boxes, polygons, etc.

"sw" allows the draw routines to discover that another menu item has been selected, to know that it is time to stop the current processing, and to proceed next as appropriate, via "type".

The Main Program Body

Here, the routines within the main program body are executed as a direct result of the menu selection (via the "type" variable).

If it's appropriate to wait for the next menu selection, "type" directs the program to "WaitRtn" which uses the SLEEP statement to wait for the next menu selection. (Note that SLEEP may also be used to wait for MOUSE events, if you have also included a MOUSE interrupt routine within your program).

If some form of processing is required, the processing routine only maintains control until another menu item has been selected, (via "WHILE sw=0 ... WEND"). If you remember, the variable "sw" is set to zero just before the routine is given control; and the only way it can be changed is via the menu interrupt routine. In this example, it is true for the routines that draw the various patterns on the screen; they simply keep executing until "sw" changes value!

You will undoubtedly notice that the only way "QuitRtn" can be reached is when the user selects "Quit" under the "Run" menu, (and only after they have closed the other windows at that)!

On Interrupt Routines in General

Now, if you are not used to the concept of interrupt routines, this may all seem a little strange, but study the program carefully. It is really very simple (though a little different), and I'm sure you will get the hang of it very quickly.

It is important for two reasons. First, interrupt routines may also be used to handle mouse "clicking"; and secondly, to gain control whenever an error occurs. You should already be beginning to see the possibilities of using interrupt routines in your programs.

However, if you are still having problems seeing their importance, consider the operating system that is driving your Amiga. It functions almost solely on interrupts; for one thing, that's how it handles multitasking. Each time an interrupt (and the possible interrupts that it may encounter far exceed anything we have discussed here) occurs, it gives control to the next task on its "task list". This task then maintains control until the next interrupt occurs, when the next task receives control, and so on as long as there are tasks to service!

Understanding them is important because while they will not be needed in many programs, occasionally you will be able to make really effective use of them.

An Extension Exercise

As a first exposure to using interrupts, why not try a simple expansion to this program? Once you understand how it functions, just add a third window in which to draw circles and ovals! The code necessary to do this is, for the most part, a duplication of code that already exists; (the polygon code is essentially a duplication of the boxes code). While not essential, one thing you should do is to make the "Circles" menu the third one, and move the "Run" menu to position number four. (Remember, the menu is a direct reflection!)

A couple of final notes. When using custom menus, it is not mandatory to use an interrupt routine. There is nothing wrong with using "x=0:WHILE x=0:x=MENU(0):WEND" to wait for a menu selection. In some cases it may even be the most appropriate method; but it does not provide a great deal of flexibility, it frequently requires a lot of duplicated code, modifications will become increasingly complicated and CPU cycles are being used continually.

There are also a couple of other statements that may be used when processing custom menus which we have not discussed. They simply provide even more flexibility, and you may find out about them in your Amiga Basic manual.

Well, I trust you have found this little excercise interesting, and that you are now familiar enough with custom menus and interrupt routines to try them in your own programs. They do require a slightly different thought process (if you use the interrupt approach) in the design of your programs, but they provide a tremendous amount of flexibility. They also remove the need for you to supply your own error checking and handling code - it's all built in!

One more door has been opened!

```
' Amiga Basic Menu Demonstration Program
' Version 1.0   by Bryan D. Catley

'CLEAR:RANDOMIZE TIMER
SCREEN 1,320,200,4,1
WINDOW 2,"Menu Examples:",,16,1
COLOR 3,2:CLS
MENU 1,0,1,"Boxes"
MENU 1,1,1,"Open"
MENU 1,2,0,"  Square"
MENU 1,3,0,"  Oblong"
MENU 1,4,0,"Clear"
MENU 1,5,0,"Close"
MENU 2,0,1,"Polygons"
MENU 2,1,1,"Open"
MENU 2,2,0,"  3 Sides"
MENU 2,3,0,"  Random"
MENU 2,4,0,"Clear"
MENU 2,5,0,"Close"
MENU 3,0,1,"Run"
MENU 3,1,1,"Quit"
MENU 4,0,0,""
LOCATE 7,6:PRINT"P r o g r a m  C o n t r o l"
LOCATE 9,15:PRINT"i s   v i a"
LOCATE 11,16:PRINT"M E N U S"
LOCATE 16,8:PRINT"(Use Right Mouse Button)."
MENU ON:ON MENU GOSUB MenuInt
opncount=0:type=2

' Perform Routine Necessary as a Result of Menu
Slection '
DoMenu:
sw=0
ON type GOTO
QuitRtn,WaitRtn,BoxesRtn,BoxesRtn,PolyRtn,PolyRtn

WaitRtn:
SLEEP:GOTO DoMenu

BoxesRtn:
WHILE sw=0
  windwid=WINDOW(2):windhit=WINDOW(3)
  bx1=INT(RND*windwid)-1:bx2=INT(RND*windhit)-1
  x=windwid-bx1:y=windhit-bx2
  col=INT(RND*15)
  IF type=3 THEN
    IF x>y THEN x=y
    bx3=INT(RND*x)
    IF sw=0 THEN LINE(bx1,bx2)-STEP(bx3,bx3),col,bf
  ELSE
    bx3=INT(RND*x):bx4=INT(RND*y)
    IF sw=0 THEN LINE(bx1,bx2)-STEP(bx3,bx4),col,bf
  END IF
  FOR n=1 TO 500:NEXT
WEND
GOTO DoMenu

PolyRtn:
WHILE sw=0
  windwid=WINDOW(2):windhit=WINDOW(3)
  IF type=5 THEN n=3
  IF type=6 THEN n=INT(RND*5)+3
  FOR m=1 TO n
    x=INT(RND*windwid)-1:y=INT(RND*windhit)-1
    IF x<1 THEN x=1
    IF y<1 THEN y=1
    AREA (x,y)
  NEXT
  IF sw=0 THEN COLOR INT(RND*15):AREAFILL
  FOR n=1 TO 500:NEXT
WEND
GOTO DoMenu

QuitRtn:
MENU OFF:MENU RESET
WINDOW CLOSE 2:SCREEN CLOSE 1
END
```

```
' This is the Menu Interrupt Routine. It
Automatically
' Receives Control when a Menu Selection is Made
'
MenuInt:
menu0=MENU(0):menu1=MENU(1)
ON menu0 GOTO MIBoxes,MIPolys,MIRun

MIBoxes:
ON menu1 GOTO MIBxOpn,MIBxSq,MIBxOb,MIBxClr,MIBxCls
MIBxOpn:
MENU 1,1,0:MENU 1,2,1:MENU 1,3,1:MENU 1,4,1:MENU 1,5,1
WINDOW 3,"Boxes:",(8,8)-(232,120),22,1
wind3col=INT(RND*15):COLOR ,wind3col:CLS
opncount=opncount+1:IF opncount=1 THEN MENU 3,1,0
type=2:GOTO MIExit
MIBxSq:
MENU 1,2,2:MENU 1,3,1
WINDOW 3
type=3:GOTO MIExit
MIBxOb:
MENU 1,2,1:MENU 1,3,2
WINDOW 3
type=4:GOTO MIExit
MIBxClr:
MENU 1,2,1:MENU 1,3,2
WINDOW 3:COLOR ,wind3col:CLS
type=2:GOTO MIExit
MIBxCls:
MENU 1,1,1:MENU 1,2,0:MENU 1,3,0:MENU 1,4,0:MENU 1,5,0
WINDOW CLOSE 3
opncount=opncount-1:IF opncount<1 THEN MENU 3,1,1
type=2:GOTO MIExit

MIPolys:
ON menu1 GOTO
MIPyOpn,MIPy3Side,MIPyRnd,MIPyClr,MIPyCls
MIPyOpn:
MENU 2,1,0:MENU 2,2,1:MENU 2,3,1:MENU 2,4,1:MENU 2,5,1
WINDOW 4,"Polygons:",(48,56)-(296,168),22,1
wind4col=INT(RND*15):COLOR ,wind4col:CLS
opncount=opncount+1:IF opncount=1 THEN MENU 3,1,0
type=2:GOTO MIExit
MIPy3Side:
MENU 2,2,2:MENU 2,3,1
WINDOW 4
type=5:GOTO MIExit
MIPyRnd:
MENU 2,2,1:MENU 2,3,2
WINDOW 4
type=6:GOTO MIExit
MIPyClr:
MENU 2,2,1:MENU 2,3,2
WINDOW 4:COLOR ,wind4col:CLS
type=2:GOTO MIExit
MIPyCls:
MENU 2,1,1:MENU 2,2,0:MENU 2,3,0:MENU 2,4,0:MENU 2,5,0
WINDOW CLOSE 4
opncount=opncount-1:IF opncount<1 THEN MENU 3,1,1
type=2:GOTO MIExit

MIRun:
ON menu1 GOTO MIRunQuit
MIRunQuit:
type=1:GOTO MIExit

MIExit:
sw=1:RETURN                                        •AC•
```

IFF Brush to AmigaBASIC 'Bob' Editor

By Michael Swinger

I bought my Amiga primarily for the graphics and animation possibilities, as I suspect many others did, and planned to write my own applications in BASIC. When Amiga Basic was finally released in December, it was encouraging to see that most of the graphics potential could be tapped in BASIC. On the other hand, it was discouraging that the supplied Object Editor was capable of creating only small, primitive shapes. It proved difficult to rewrite.

Deluxe Paint was released at about the same time, and the first thing I did was to try to stick an IFF file created in Deluxe Paint straight into the BASIC OBJECT.SHAPE statements - which didn't work, of course. If you LIST the Object Editor program, the very first page tells you the format of the file it produces. When the IFF format was made public it was then possible to see why the two file formats were incompatible, but it was also clear that they both contained much the same information, and that it should be possible to convert one format into the other.

The listing here is a graphics editor in Amiga Basic. It will convert Deluxe Paint files into the same format as that produced by the BASIC Object Editor, so that a "brush" created in Deluxe Paint can be programmed as a "BOB" in AmigaBasic, using the OBJECT.SHAPE commands.

I wrote the editor for my own use. It has several limitations: One, the brush file cannot be compressed, so that the maximum size of a bob is around 40 pixels wide by 90 pixels tall. If the brush is larger, the bit map will be compressed, since it takes up too much space. Two, although the brush file contains the color information for the PALETTE statements, it will not be written out.

So that you can refine and improve this editor, if you wish, let me describe briefly the format of an IFF brush file (such as that produced by Deluxe Paint, Aegis Images, or Graphicraft), and the AmigaBasic file handling routines you will have to grapple with.

You can examine a binary file in the CLI by entering TYPE <filename> OPT H. The hexadecimal representation of the data in the file will be arranged in four groups of digits on the left of the screen: the right side shows any ASCII equivalent. A Deluxe Paint brush, the "Wizard", looks like this:

464F524D - This spells "FORM", An ID.

00000A24 - The size of the file, stored in eight hex digits, which is 4 bytes, also known as a 'long word'.

494C424D - Spells "ILBM", Interleaved Bitmap--this is what will create most of the incompatibility problems--more on this later.

424D4844 - Spells "BMHD", Bitmap Header, another ID.

00000014 - The size of the header information which follows.

0043003E - First four hex digits are width of the image; the second four are the height. This is stored as 2 'short words' of 2 bytes each.

00000000 - X and Y origin for grabbing a brush.

05020100 - The first pair of digits (a single byte) is the depth, # of color planes; the second byte is the transparency or background register; the third byte is the compression type--a 0 means the file is uncompressed, a 1 means the bitmap has been compressed--more trouble; the fourth byte, which should be 0, is merely padding to insure that everything is beginning on an even byte boundary.

00000A0B - The first short word indicates which bit pattern means "transparent"; the second short word is the pixel aspect ratio, in this case 10:11, or 320 X 200.

014000C8 - 2 short words which give the width and height of the page.

434D4150 - Spells "CMAP", the ID for the color map.

00000060 - The size of the CMAP which follows.

The CMAP stores the color data as triplets of red, green, and blue for each color register in values of zero to fifteen. The first two long words hold the RGB values for the first two registers and part of the third in this example:

00000D/F0 F0F0/D0D0

The RGB values would be 0,0,13; 15,15,15; etc. Divide these values by 15 for an approximation of the PALETTE values to be used in AmigaBasic. The problem is that the information is stored in each byte in reverse order--ie., F0 instead of 0F. The CMAP continues until all the registers are described.

At this point you will encounter another series of IDs, depending on whether the file is of a brush or a full screen. If the bytes spell "GRAB", this is for a brush. If it spells "CRNG", this is the color cycle information for a full screen image.

424F4459 - This spells "BODY", the actual bit map of the image.

00000988 - The size of the BODY which follows.

For some reason the body of an IFF bit map is interleaved, which may make it easier for a human to inspect the data. Interleaving means that the first row of the image is written out for each plane, and then the next row for each plane, etc. We have to "de-interleave" the data so that an entire plane is written out for the whole image, row by row, before going on to the next plane.

If you have LISTed the Object Editor (or typed LLIST in the output window to get a printout of the program) you can see that the IFF data simply has to be read in to an editing program, filtered to extract only the data we need, and then re-arranged to be written back out in a string format that the OBJECT.SHAPE statements can use.

The CVL function will read a long word (4 bytes) from a file and convert it into an integer. The MKL$ function is the mirror function to write it back to a file in string format. CVI and MKI$ do the same for short words of 2 bytes. To convert a single byte to an integer, the ASC function works just fine. The editor will prompt you for the complete pathnames of the files to be converted and the files to be created, so keep notes of these. You can avoid lots of frustration by specifying the pathname of a file by its drive, rather than its disk name, especially if you're using one drive. When you first create the brush in whatever graphics program you are using be sure to use the first color as the background color, and specify that color for PALETTE 0 in your BASIC program so that the same color ends up as the transparent background for your bobs. Some refinements you can add to the editor are routines to decompress files, write the CMAP data into a series of

PALETTE statements, and the ability to write the data for bobs or for the PUT and GET routines.

Program listing:

```
' **  Instructions, sort of  **
' This editor will convert a BRUSH created in
Deluxe Paint
' into the format required for a BOB by the
OBJECT.SHAPE
' statements in AmigaBasic.
' The brush can be in 320 or 640 X 200 or 400
resolution
' and in 32, 16, or 8 colors.
' At the moment the Deluxe Paint file cannot be
compressed.
' An approximate safe  max. size is 40 pixels wide
X 90
' pixels tall--
' An "input past end" error when loading the IFF
file means
' you're trying to read a compressed file.
' Graphicraft (v. 1) doesn't compress files, so
this editor
' could read a full screen, if you can then figure
out how to
' stuff the resulting 41 K or so into a BASIC array
to PUT it
' on the screen.....
' If you want more information on BOB and SPRITE
file
' formats, list  the Object.editor program in the
demos on the
' EXTRAS disk.  For  more info on IFF files, see
all the stuff on
' Compuserve, especially a file 'called "IFF.text",
which
' describes the file format.
' Fiddle with this, please!! It needs a
decompression routine!
'       --Mike Swinger  6/28/86  Columbus ASIG

CLEAR, 30000&
SCREEN 1,640,200,2,2
WINDOW 1,"IFF.EDITOR",,15,1

START:
DIM BODY$(5,200)
CLS:BEEP
PRINT "** PLACE THE DISK CONTAINING THE FILE YOU
WANT TO EDIT IN THE DRIVE"
PRINT "** AND TYPE IN ITS COMPLETE PATHNAME, AS IN
THE FOLLOWING EXAMPLE"
PRINT
PRINT "        DF0:BRUSH/WIZARD            "
PRINT
INPUT "** PRESS  <RETURN> WHEN FINISHED --->
",FILENAME$
OPEN FILENAME$ FOR INPUT AS 1

READ.FILE:
JUNK=CVL(INPUT$(20,#1))
WYDTH=CVI(INPUT$(2,#1))
HEIGHT=CVI(INPUT$(2,#1))   ' WIDTH &HEIGHT IN
PIXELS
JUNK=CVL(INPUT$(4,#1))
PLANES=ASC(INPUT$(1,#1))        ' # OF BIT-PLANES
MASK=ASC(INPUT$(1,#1))          ' THE TRANSPARENCY
```

```
REGISTER
COMPTYPE=ASC(INPUT$(1,#1))      ' COMPRESSION TYPE
JUNK=ASC(INPUT$(1,#1))
JUNK=CVI(INPUT$(2,#1))
HASPECT=ASC(INPUT$(1,#1))
VASPECT=ASC(INPUT$(1,#1))     '  RESOLUTION
 IF PLANES=5 THEN
 JUNK=CVL(INPUT$(124,#1))
   ELSEIF PLANES=4 THEN
   JUNK=CVL(INPUT$(76,#1))
    ELSE
    JUNK=CVL(INPUT$(52,#1))
END IF
SIZE=CVL(INPUT$(4,#1))
ROWBYTES%=INT((WYDTH-1)/16+1)*2 'This is the hard
part
 FOR I=0 TO HEIGHT-1                    'that Lee
Savory helped with
  FOR J=0 TO PLANES-1
   BODY$(J,I)=INPUT$(ROWBYTES%, #1)
  NEXT J
 NEXT I
CLOSE #1

PRINT.SPECS:
XC=2^INT(PLANES)-1 : REM TOTAL# OF COLORS
IF INT(HASPECT)=10 THEN RES=320 ELSE RES=640
PRINT "        YOUR FILE:  "; FILENAME$ :PRINT
PRINT " WIDTH IN PIXELS: ";WYDTH,
PRINT "HEIGHT IN PIXELS: ";HEIGHT :PRINT
PRINT " # OF BIT-PLANES: ";PLANES,
PRINT XC+1; " COLORS" :PRINT
PRINT " THE RESOLUTION IS "; RES ;: PRINT " X  200
"
PRINT " THE SIZE OF THE BODY IS"; SIZE
PRINT:PRINT

WRITE.FILE
PRINT " PUT DISK IN DRIVE AND TYPE COMPLETE
PATHNAME OF FILE YOU WANT"
PRINT " TO CREATE       (EXAMPLE--DF0:BLOB  "
INPUT " PRESS RETURN WHEN FINISHED ---> ",FILEOUT$
OPEN FILEOUT$ FOR OUTPUT AS 2
PRINT #2, MKL$(0);
PRINT #2, MKL$(0);
PRINT #2, MKL$(PLANES);
PRINT #2, MKL$(WYDTH);
PRINT #2, MKL$(HEIGHT);
PRINT #2, MKI$(24);
PRINT #2, MKI$(XC);
PRINT #2, MKI$(0);
 FOR J=0 TO PLANES-1
  FOR I=0 TO HEIGHT-1
   PRINT #2, BODY$(J,I);
  NEXT I
 NEXT J
CLOSE #2

INPUT " TYPE <1> TO EDIT ANOTHER FILE, OR <2> TO
QUIT ",S
ON S GOTO RESTART, QUIT

RESTART:
    CLEAR : GOTO START

  QUIT:
END
```

•AC•

2 MEGs For Your *AMIGA!*

A must for software developers
Allows more programs to run simutaneously and faster
Can be used to increase system RAM and/or as a FAST RAM DRIVE
Uses standard memory bus architecture to allow future compatibility
Allows full use of memory expansion port for additional peripherals
In Use By Many Major Software Developers For Over Six Months

AX2000 2 MEG RAM Board $899.00U.S. ($1276.00 CDN)
AX1000 1 MEG RAM Board $729.00 U.S. ($1035.00 CDN)

Complete in case, nothing else to buy!
1 year manufacturer warranty!

DEALER INQUIRES INVITED

Comspec Communications Inc.
153 Bridgeland Avenue, Unit 5
Toronto, Ontario, Canada
M6A 2Y6 (416) 787-0617

Shipping via courrier: within Canada add $25.00. To U.S.A. add $100.00 U.S. - includes custom clearance

The AMICUS Network

By **John Foust**
Compuserve [72337,135]
People Link AMICUS
Delphi JOHNFOUST
uucp through the Well "jfoust"

I walked into an Amiga dealer in another city. The storefront window had a prominent Amiga banner, and a shelf and table near the window had the most Amiga software that I had ever seen in one place, including COMDEX. Best of all, there was a healthy stack of the latest issue of Amazing Computing.

A box for True Basic caught my eye. At this time, the demo disks were just shipping, and I didn't think the package was out yet. I was pleasantly surprised. This store is in a not-so-large northern Wisconsin town. I asked about the magazine. The salesman said it was selling well.

He was thinking about sending for the AMICUS disks, but he hesitated, since one customer brought in AMICUS disk 9, and he felt it mightn't be worth the six dollars. He worried that their was a lot of duplication between the Fred Fish disks and the AMICUS disks.

I hope this hesitancy isn't common among dealers. Six dollars a disk isn't much, considering it takes someone two minutes per disk copy, and more time to address and stamp the envelope. Ordering a collection of twenty disks is a formidable expense. If you are unsatisfied, you can always reformat the disks. The consumer price of blank Amiga disks is four to five dollars each, in a box of ten, at many dealers, although the smart shopper gets them mail order.

Duplication is a justified thought, of course. Hey, I must have 200 disks of Amiga software, and I've given up all hope of organizing it. However, I have tried to keep duplication to a minimum. There will always be a small amount, since I don't know the contents of the Fish disks in advance. Also, Fred Fish claims to "speak no BASIC", so I think he shies away from AmigaBasic, and favors hard-core C programs.

I think the AMICUS disks are the best value in the Amiga software market today. It's certainly much cheaper than subscribing to a commercial network, and downloading software. The disks are well-organized, and most function in a one-drive system, and 99 percent of the texts and programs are accessible from Workbench. Some dealers reorganize them for customer satisfaction. For instance, they might make a telecommunications disk to help sell modems, or an Amiga Basic disk to promote a user group.

The Lure of Large Memory

It is hard to confess, because I know many Amiga owners will begin to lust, but I've had a two-and-a-half megabyte Amiga for several months. I can't imagine going back to 512 K memory. Extra memory truly spoils the Amiga owner.

Picture this: No more 'brrmps' or 'gronks.' No requests for the Workbench disk. Imagine 1.8 megs free on the Workbench memory indicator. Imagine compiles and links without disk access, and at much greater speed.

It comes for a price, of course. The Comspec board I've been using has a list price of $1079 American, but the lack of custom fees in Canada brings this to about $900 there. I'm sure memory board prices will drop, as the market widens, and more people join the Amiga fold. So far, there are at least a half-dozen other contenders in the RAM expansion market.

Ah, the joy! The absence of 'gronk, gronk, gronk' makes it worthwhile for me. The increased speed is a joy, too. The RAM disk operates much faster than the disk drives. Version 1.2 of the operating system increases the speed of the RAM: disk even more.

In my Workbench startup-sequence, I first start a new CLI process. This CLI window lets me start my work immediately, while the startup-sequence finishes in the initial background CLI.

The next line in the batch file is the RAM test and the Comspec 'AddMem' program. It takes a few seconds to check the memory, and tell the operating system that this memory can be used for programs.

'AddMem' is the program supplied on the developer disks, for use with the prototype memory expansions that Amiga used in-house. Each memory expansion box on the market comes with a program like this, if the original 'AddMem' doesn't work, or if they want to add a custom memory check program, as Comspec did. Their program also performs the function of 'Addmem.'

Reportedly, some boxes will 'auto-config', which means the operating system will recognize the board immediately, and add the memory itself. I'd suspect these 'autoconfig' boards will cost extra, since it requires more chips and design effort. This auto-configuration ability is a new feature of version 1.2 of the operating system, due out later this fall.

Next, the startup-sequence copies most of the Workbench disk to the RAM: disk. This happens in the background, while I'm doing something else in the CLI. This takes about two minutes, as compared to the 15 second boots without the RAM expansion. Then it loads the Workbench.

I must admit, I don't use the Workbench alot. When my Amiga had only 512K, I didn't even load the Workbench, because I would rather use the memory for a RAM: disk. Now, with several megabytes, who cares? After the initial gronking, I won't have another disk access, unless I ask for one explicitly. A series of ASSIGN commands in the batch file tell the

operating system to look at the files in RAM:, instead of the files in DF0:, whenever I type a command in the CLI.

The RAM: disk is an easy place to store things. It's like a hard disk, everything is there, and you don't have to worry if you've stored it on the right disk.

Download to RAM:

When downloading or uploading files from a network, I use the RAM: disk. My terminal program, Online!, normally writes out each block of data as it is received. This can waste a lot of time, and wasting time on a pay-by-the-minute network is not my idea of fun.

With the RAM: disk, uploading or downloading happens much faster. Of course, this works well on a 256 K or 512 K Amiga, too. If you say "RUN ONLINE!" in the CLI instead of just "ONLINE", you can switch back to the CLI, and copy each downloaded file to disk, while your terminal program is still running. To free up memory, you can delete each file after it is copied to disk, while another file is downloading to the RAM: disk.

In the RAM: disk, the files cause no conflicts with what I'm doing on the floppy drives. If the upload or download is going to take a while to complete, I can click the telecom window to the back. A beep signals the end of the transmission. Meanwhile, I can work on something else.

Pain Squared!

If you've ever been angry after a visit from the Guru, consider amplifying that anguish several times with a RAM: disk. Of course, you are lulled into a sense of security after several days of no trouble. If you've got several files in RAM:, and a program crashes, you lose the files. Perhaps you've had this happen already, but what if you had a megabyte or two of files in the RAM: disk? Rebooting takes a long time, so you won't be back up to speed for several minutes, and the files are gone, daddy, gone.

One night, I crashed the machine while ARCing files for uploading to a network. In the background, I was uploading a large file, and it had been going for about ten minutes. I lost about three hour's work, all told. I was furious! I still had copies of the files in question, but ARCing takes a long time, and the ARC files would just clutter my disks, so I left them in RAM:.

The crash was caused by a program I found on a public domain disk I got in the mail. Needless to say, I erased the program after this, and formatted the disk to further curse its bits.

Now, everytime I run a new program, I save the important files in RAM: to disk. In general, I've been more careful, and I save my important RAM: files to a spare disk every quarter hour.

When compiling or assembling, I don't have a single disk access, and it goes so much faster. The Lattice compiler, the Aztec compiler, the Metacomco assembler, and the object libraries all fit into RAM:, with room to spare for my source code. I truly think I could get along with one drive, if I didn't have to copy disks so often.

It almost makes the Amiga a nice development machine. Let's face facts, program development without a hard disk is a hassle. The combination of a large hard disk and a large RAM disk is almost ideal. Floppies and limited memory are an frustrating pair of limitations. I can't wait to get a hard disk, to shorten the time it takes to boot, and to say goodbye to gronks and disk swapping forever.

At first, I was inclined to run as many programs as possible at the same time. I soon noticed a degradation in speed. My telecom program would display jerky bursts of characters, and the disk drives would gronk at a different frequency. Mouse and window re-sizing speed would drop.

Some programs don't work well with the extended memory. The most common problem shows up in gadgets in the program. Instead of a detailed drawing of a button or switch, the gadget is solid black, or filled with random bits.

A proper Amiga program must ask the operating system for memory visible to the Amiga graphics chips. The RAM expansion on the front of your Amiga is visible to the graphics chips, but the memory expansion on the side is not.

The operating system calls the graphic memory chip memory. The programmer must specifically request chip memory for the renderings of gadgets, otherwise, the drawings will not be displayed correctly.

Dan 'DJ' James, a regular on People Link, wrote a program that adjusts program files so this problem is removed. An Amiga executable file is composed of several pieces, and the operating system loads these pieces into the type of memory needed for the program, in chip or non-chip memory.

James' program massages the program file, and tells the operating system to put all data objects in chip memory. This program works well, and is on one of the newer AMICUS disks. See the catalog for more information.

Future Amiga chips

Confirming long-standing rumors, Jay Miner, the general manager at Commodore-Amiga, announced that C-A plans new Denise and Agnus chips for future Amiga computers.

Agnus is the video chip in the Amiga. The current chip can only address 512K of memory, meaning that all sampled sounds and video graphic images must reside in this memory, also known as 'chip' memory.

The new chip can manipulate 2 megabytes of memory. The new chip set will also support non-interlaced graphics. We'll have more details in a future issue. The news arrived at press time.

Modula-2 from Switzerland

The latest news on the Modula-2 compiler on Fred Fish disk 24 came across Usenet recently. According to Claudio Nieder, of the computer science department at ETH Zurich, this version on the Fish disks is a pre-release of the

Modula-2 Compiler used for MacMETH for the Macintosh. A new version and system will be available soon. Hopefully, 'soon' means something different in the Swiss software vocabulary.

"The current version of the Compiler, alpha tests are coming soon, works much better than the PD version, also the run-time system is much better. Range errors and other Traps don't lead to Gurus anymore," he said. "The system will be release at the end of this year and will include all necessairy interfaces, a source-code debugger, a linker..." Nope, it sounds like Wirth's grad students understand the American software industry, if 'soon' means five to six months.

Did you know that Americans call Prof. Wirth by value, and that the Europeans call him by name? Did I tell this joke already? You see, we pronounce it 'worth', and they say 'veerth', the way he pronounces it.

New AMICUS disks

Disk 11 is finally ironed out. First, here are the C programs on the disk. As I mentioned two issues ago, it has a program called 'pm', a performance monitor. It presents a chart-recorder-type display, in a sizable window. It displays a rolling chart of CPU activity and memory use, separated into chip and fast memory.

'ps' is like the Unix command of the same name. It shows a list of the currently active CLI processes, and information about them. By Dewi Williams.

'cpri' shows a list of CLI processes, and lets you adjust the priority of a process. In a multitasking computer, each program in memory gets a share of the processor's time. The priority is a measure of that share. By adjusting the priority of a process, you can speed the execution of one process, at the sake of the others.

'vidtex' is a program to display the run-length-encoded pictures from Compuserve. For example, up-to-the-minute weather maps are available in one forum on Compuserve. With any terminal program, you can capture the graphic weather display. It is sent as a stream of seemingly meaningless characters, but this program will turn it into a high or medium resolution picture. This was written by Amazing Computing author George Musser.

This disk contains several programs from Amazing Computing. It is much easier to get this disk than to type in all the source code to a multi-page Amiga Basic program.

Author Steve Pietrowicz has two contributions on this disk, including the pointer editor article in this issue, and his example programs from the discussion of optimization of Amiga Basic.

There is an extensive calendar, diary and date book program, written by Mark Hurst, of Sheridan, Oregon. In a strange connection, this program came to me through a pitching coach of the Seattle Mariners. It is superbly done. For example, when you open the diary, the pages are animated, and the cover of the book opens when you start.

The IFF brush to Amiga Basic BOB converter program, by author Jim Swinger, is here. BOBs are the OBJECT shapes you can draw in Amiga Basic.

Jim Meadow's 3D graphics program is here. There is a loan amortization program, a program to draw and play sound waveforms, a program to draw Hilbert curves, a 'madlib' generator program, a talking mailing list program, a program to demonstrate reading the mouse on a high resolution screen, a slot machine game, a strange, pachinko-like game, a game of TicTacToe, and a short program that makes wierd sounds.

For executable programs, there are three programs for displaying the three most popular Commodore 64 paint program pictures. One displays Doodle! pictures, one shows Koala Pad pictures, and one shows pictures.

There is a program called 'cp', like the Unix command. This is a copy program, like the AmigaDOS copy command, except it uses the popular '*' wildcard filename method of Unix, CP/M and PC-DOS.

'diff' and 'ssed' are more Unix-like programs. 'diff' compares two text files, and outputs a list of their differences. 'ssed' is a stream-oriented editor that can take the output of 'diff', apply it to one of the files, and produce the other.

'dirutil' is a handy, Intuition method of manipulating files on a disk. You can delete, rename, copy and move files, all within Intuition-style menus and requesters.

A single assembly language program, 'cls', by Tom Caldwell, is a short example of sending a clear-screen sequence to the screen, and handling simple command line arguments.

Modula-2 is more and more attractive as an Amiga development language. Like assembly language, development is hampered by a lack of examples, and the C-language emphaisis in the ROM Kernal Manual.

TDI Modula-2 has a recent bug fix that corrected a long list of problems in the original release. Les Caudle, of TDI, claims the new version is much faster, and produces smaller, faster code than before. He said the Sieve benchmark is now down to about 2400 bytes of executable code.

Richie Bielak was a major bug-slayer and Modula guru in this effort. Three of his example programs are collected here. One is called 'trails', which is one of those moving worm displays. Another is a simple requester example, and another is a lower-to-uppercase converter program, a handy tool for sloppy typists.

There is a single Forth example of a well-known circle-drawing algorithm. This short example is by Amazing Computing columnist Jon Bryan.

Last, there is a shareware collection of spreadsheet templates for the Analyze! spreadsheet, from Micro-Systems Software. There are a dozen here, they include a checkbook balancer, an inventory sheet, and a loan amortization sheet.

New Fred Fish Disks
The Fish disks are now 30 in number.

Disk 25 contains a new version of Hack, ported to the Amiga by John Toebes. This is the graphic version, as compared to the character-based graphics of the previous version.

Disk 26 has 'unhunk', a program to process the Amiga executable load file format. It lets you coalesce code and data hunks. 'ckermit' is a nearly complete Kermit server, with a connect mode. 'ps' is the process priority program described in AMICUS disk 11. 'archx' bundles a series of text files for transmission through a network.

Disk 27 has AmigaBasic demos I promised on AMICUS disk 13. These include the new ConvertFD program, and programs to use the new IFF hunk type called ACBM, which is a format more conducive for display under Amiga Basic. ScreenPrint is an example of using library calls to access AmigaDOS functions.

Disk 28 has more ABasic games, by the author of the Monopoly game. These include cribbage, backgammon, mille bournes, and othello. For C programs, this has 'shar', the Unix-compatible file archiving and unpacking program, and SuperBitMap, which demonstrates the use of ScrollLayers, and how to sync SuperBitMaps for printing.

Disk 29 contains a demo version of Aegis Draw. The Save feature is disabled, but otherwise, this is the real production program. 'cc' is another version of the C compiler front-end program from an early Fish disk, but this version is set up for the Manx compiler.

'enough' is a program to test for the existance of a given resource such as memory, disk drives, etc. from the CLI.

'player' is a public domain playing program for Aegis Animator script files.

'rubik' is an animated Rubik's cube program, a merger of the 'skewb' and 'amiga3d' programs.

This disk has a public domain implementation of a C string library.

'vt100' is a terminal program that emulates the VT100 terminal. It also has Kermit and Xmodem protocols.

Disk 30 is a collection of shareware programs. It has the Amiga Basic BBS I promised for AMICUS disk 12.

FontEd is an Amiga font editor, which might hold you until the official font editor comes with version 1.2 of the operating system.

MenuEditor is a useful program for programmers. It lets you design Intuition menus, and then outputs the proper C code to effect those menus in your C program. Mostly, this is a translation of the things you designed into data arrays in the right format for the operating system. This sort of program is sorely needed for the Amiga. Most other windowed operating systems rely on a program like this, such as the Macintosh, or Windows on an IBM PC.

StarTerm is a program by one of the SYSOPs of Compuserve. This is a full-featured terminal program that exceeds the capabilities of Online in some ways.

Please note that these programs are shareware. This means the author is requesting money from you, if you and your conscience feel that you get sufficient benfit from its use, they want your money. Since these authors are more sensitive to the 'public domainness' of their property, the contents of this disk might prevent Amazing Computing from distributing it.

This is the addendum to the AMICUS list, to delineate AMICUS 11.

Disk 11
C programs

dirutil	Intuition-based, CLI replacement file manager, S-E
cpri	shows and adjusts priority of CLI processes, S-E
ps	shows info about CLI processes, S-E
vidtex	displays Compuserve RLE pictures, S-E

Amiga Basic programs

pointered	pointer and sprite editor program
optimize	optimization ex ample from AC article
calendar	large, animated calendar, diary and date book program
amortize	loan amortizations
brushtoBOB	converts small IFF brushes to Amiga Basic BOB OBJECTs
grids	draw and play waveforms
hilbert	draws Hilbert curves
madlib	mad lib story generator
mailtalk	talking mailing list program
meadows3D	3d graphics program, from Amazing Computing article
mousetrack	mouse tracking example in hires mode
slot	slot machine game
tictactoe	the game
switch	pachinko-like game
weird	makes strange sounds

Executable programs

cp	unix-like copy command, E
cls	screen clear, S-E
diff	unix-like program to show differences between files
ssed	unix-like stream editor uses 'diff' output to fix files
pm	chart recorder performance indicator

Assembler programs

cls	screen clear and CLI arguments example
Modula-2	trails moving-worm graphics demo
caseconvert	converts Modula-2 keywords to uppercase
Forth	Breshehan circle algorithm example
Analyze	12 templates for the spreadsheet Analyze!

•AC•

LINKING C PROGRAMS WITH ASSEMBLER ROUTINES ON THE AMIGA

By **Gerald Hull**
People Link DRJERRY

One of the more common approaches to software development is to code the application initially in a high-order language (HOL), to get it up and running as soon as possible. Then, as necessary or desired, various subroutines are rewritten in assembly language for increased speed and efficiency.

Whatever your motivation -- you may have some particular algorithms already coded in assembler, or wish to perform low-level functions that aren't easily accomplished otherwise -this article will show you how to interface routines written in 68000 assembly language with an Amiga HOL.

The particular HOL in my example is Lattice C, and the specific interfacing procedure I describe is pretty much limited to that language, insofar as it relies upon use of "Alink."

Manx C would use a similar method of linking assembly language routines. It has a different frame pointer, and its registers are organized differently than Lattice. Of course, in Manx, in-line assembly code is permitted. In TDI Modula 2, you can insert in-line code, in the form of constants, but not true assembly language source text.

However, so far as my description of HOL structures goes, much of what I say should carry over to other dialects of C, as well as any similar high-level languages on the 68000. Such languages use "local stack frame" (LSF) based subroutines. They include Modula-2 and Pascal, for instance, but do NOT include languages such as Basic, Forth, and Fortran. I touch on some of the variations in HOL implementation in later sections.

A "local stack frame" is a nifty way of providing any subroutine in a program with its own local variable space, located by reference to a special "frame pointer". More importantly, the LSF approach guarantees that any such subroutine can call any other and be confidant that parameter values will not get messed up, and the program's flow of control will not become confused. Such routines can even call themselves, and for this reason are said to be "recursively reentrant".

The instruction set of the 68000 microprocessor in the Amiga has several instructions designed to implement the local stack frame concept. Later on, we will examine exactly how they work. First we must confront a more immediate problem: How do we isolate the C code that we want to replace with assembler? In what remains, I assume familiarity with the Amiga's command line interface, the CLI, and ED editor, as well as access to the Lattice C compiler and the MetaComCo Macro Assembler.

I will describe the process I followed in structuring 68000 code to replace a particular C subroutine, and in assuring that it would link back into the original program.

SEPARATING A SUBROUTINE

In order to replace C code with assembly language, the first thing you want to do is isolate it in its own subroutine. If your goal is to speed up your program, try as much as possible to keep all loops internal to the routine. As we shall see, a certain overhead is accrued everytime a LSF subroutine is called.

The program in Listing 2, DOODLE.C, is nothing special: it's a computerized version of a little doodle from grade school. I doubt it is the cleverest way on the Amiga of doing what it does. However, it is relatively short, and has a routine 'doodit()' which contains a number of subroutine calls and all the serious number crunching in the program. This makes it useful for our purposes, since I designed it that way.

The next step removes the function 'doodit()' from DOODLE.C. Once it is removed, the program will not link without a reference to '_doodit'. We can play a shell game with how that is provided.

How is this done in DOODLE.C? Notice that Listing 2 contains only a single 'include' file, in the line '#include <doodle.def>'. That file, in Listing 1, holds all the other 'include' files, as well as all the '#define' statements and any "non-official" structure definitions. By putting all this information in a separate include file, I can also easily provide exactly the same information to 'doodit()' when I want to compile it separately. Further, note that the variable 'INTERNAL' has been set to FALSE in DOODLE.DEF. The symbol 'FALSE' is defined as "0" in INCLUDE/EXEC/TYPES.H.

Turning back to the listing of DOODLE.C, you will see C preprocessor directives surrounding the routine we are intending to assemble:

```
#if INTERNAL

void doodit(port, color, box)
  .
  .
  .
#endif
```

These directives provide the means by which 'doodit()' is removed from its parent program: as long as 'INTERNAL' is defined as 0 (FALSE), the C preprocessor will pass over the routine as if it isn't there.

However, not only do we want 'doodit()' out of DOODLE.C, we want it to be able to stand on its own, because the assembly language which replaces it has to stand on its own. This is why the variables it shares with other parts of the program are explicitly passed as parameters: 'port', 'color,' and 'box'. 'doodit()' also shares the constants NODE, SIDE, PART, XM and YM, requiring their separate definition in the assembler version of the routine. A more modularly designed program would pass all these values to 'doodit()' as parameters.

The routine has other features which facilitates its translation into 68000 assembler. It contains no floating point math, for instance. To use "float" or "double" variables would introduce additional complexities that are best avoided. Floating point chews up time to boot. And finally, the decremental x and y for-loops were designed with an especially sexy 68000 branch instruction in mind called DBRA.

SEPARATE COMPILATION OF FILES

I will assume that you have entered the texts of Listings 1 and 2 into appropriately named files. They are also available on the AMICUS disks, and in the data libraries of several networks.

Your C disk should contain a batch file which both compiles and links programs; the custom is to call such files something like "MAKE". When you try to compile and link DOODLE.C, you will get a "link failure" message citing '_doodit' as an "unresolved external reference".

Note that DOODLE.DEF must be in your root directory, or you'll get a lot of compiler errors as well. Check RAM: to see if it contains the file DOODLE.O. If it's not there, you probably have a line like 'delete <file>.o' in your MAKE file. You'll have to comment that line out, and recompile.

Next, using the define block and write block utilities of Amiga's ED editor, put a copy of the 'doodit()' routine into a separate file: RAM:DOODIT.C. Insert the line

```
#include <doodle.def>
```

at the very beginning of the file, just as it is in DOODLE.C. In this way, you will provide reference to the very same global definitions and constants that were required for compiling the main program. The 'doodit()' routine may not need them all, of course, but the excess cannot cause any harm.

Now exit, and execute the MAKE batch file on DOODIT.C as well. If you have done everything correctly, it will compile, but fail to link, because of unresolved external references: '_main', '_GfxBase', and '_boxinit'. In fact, you now have everything you need, but in two different places.

In compiling DOODLE.C you will have produced an object file named DOODLE.O, and in compiling DOODIT.C, you will

have produced another named DOODIT.O. To produce a complete program, all we need do is link the two together, and each will provide the other with the external references it is looking for.

LINKING SEPARATE FILES TOGETHER

To do this, it will be useful to have a special batch file just for that purpose. Make a copy of your MAKE file under another name (I call mine MAKL). Then delete (or "comment out") all the lines that have to do with compilation, and amend the link sequence so that it can deal with two separate files. This is what my MAKL looks like.

```
.key file1,file2
echo "-- linking <file1>.o with <file2>.o to <file1>."
:c/alink :lib/Lstartup.obj+<file1>.o+<file2>.o
  + library :lib/lc.lib+:lib/amiga.lib to <file1>
;delete <file1>.o      *** THESE LINES
;delete <file2>.o      *** COMMENTED OUT
date >df0:now
```

Assuming that both DOODLE.O and DOODIT.O are in your RAM: directory, you can now link them together with the following command:

```
1> execute makl ram:doodle ram:doodit
```

If everything has gone right, this will leave an executable program in the RAM: directory named DOODLE. It will behave just as if we had changed the definition of 'INTERNAL' in DOODLE.DEF to '#define INTERNAL TRUE' and had compiled it. Oh great. It looks like we've gone to all this trouble for nothing! But actually we have accomplished something very important: we have the program DOODLE.C looking to a separate file for the subroutine 'doodit()'. Now all we have to do is provide an assembler-based version of that routine for the linker, and it will be none the wiser! This, of course, is where the file MACDOOD.ASM, in Listing 3, enters the picture.

Once you have gotten a copy of MACDOOD.ASM into RAM: you will want to change disks and apply the assembler MAKE batch file to it. And again you will get error messages from the linker: '_boxinit' and '_GfxBase' are unresolved. GfxBase is a pointer used to calculate the calling addresses of Make and Draw. But, more importantly, you will have left MACDOOD.O in RAM:, an object file produced from the 68000 translation of doodit().

Finally, retrieve your version of MAKL (mine's on the C disk). After making sure that you also have DOODLE.O in RAM:, you can link the C program to the assembly language subroutine with:

```
1> execute makl ram:doodle ram:macdood
```

There now should be a new executable program in RAM: named DOODLE. Mission accomplished! Now when the program goes to execute the 'doodle()' subroutine, it will be using the assembler code listed in MACDOOD.ASM rather than the original C code. For what it's worth, the new version of 'doodit()' is 40% smaller than the C version, and the revised version of DOODLE is approximately 37% faster.

HOL SUBROUTINES IN ASSEMBLER

Now we get to the really fun part: how do you design 68000 assembly language routines to ensure that they will replace C subroutines? The essential structures involved are relatively simple; they are isolated and abstracted in Figures 1 and 2.

Figure 1: A structure for recursively reentrant, local-stack-based subroutines.

SUBROUTINE STRUCT	M68000 EXAMPLE
Allocate Locale Frame	LINK FP,#LVAL
Save Needed Registers	MOVEM.L D2-D7/SP,-(SP)
Execute Function	<code>
Restore Used Registers	MOVEM.L (SP)+,D2-D7/SP
Deallocate Local Frame	UNLK FP
Return	RTS

Figure 2: A structure for calling recursively reentrant, local-stack-frame subroutines.

Push Params. on Stack	MOVE.L 16(FP),-(SP) MOVE.L 12(FP),-(SP) MOVE.L 8 (FP),-(SP)
Save Regs. if Necessary	MOVEM.L D0-D1,LVAR(FP)
Jump To Subroutine	1JSR _BOXINT
Get Return Value	MOVE.L D0,12(FP)
Restore Any Saved Regs.	MOVEM.L LVAR(FP),D0-D1
Clean Up Stack	LEA 12(SP),SP

The first indicates the overall features of your 68000 routine. The second illustrates how to make calls to other subroutines, including Amiga ROM Kernal modules. These recipes can be adapted to one's own needs.

Let's now focus in on some of the details of MACDOOD.ASM, to see how those structures are implemented, and to illuminate other features of the 68000 code.

Perhaps the most common and most rewarding strategy for speeding up assembly code is to keep as much data as possible in registers. This works best, of course, with processors that give you many general purpose registers. Instructions work faster on register variables, and you avoid the expense of fetching and storing information from memory.

The 68000 family of processors facilitates this strategy by providing the user with a total of 16 general purpose registers. More precisely, Motorola has given us eight data registers, D0 through D7, and eight address registers, A0 through A7. Unfortunately, many assembler instructions distinguish between the two types, which qualifies the term "general purpose". On the other hand, the architecture is so much cleaner than Intel.

I have made full use of registers, as the initial comments on "registerusage" indicate. Of the address registers, A7 is always the "stack pointer", or SP. I have chosen A6 to be my "frame pointer" (FP) -- we'll be hearing more about it shortly.

Aside from these structural roles, A2 is the only other address register I use. I need it because the variable 'box' is a pointer. It contains the address of a sequence of memory locations, as opposed to what is in those locations. We will shortly learn the advantage of this.

The other registers mentioned are used more or less equivalently to variables in the C version of 'doodit()', as the comments point out. D4 through D7 are used for different variables at different times.

Listing 3 begins with a number of "assembly directives". Unlike ordinary assembler commands - for example, MOVE - they do not produce any machine code, but provide processing instructions to the assembler.

The XREF statement tells the assembler that certain labels won't be found in this particular piece of code, while the XDEF statement tells it what labels it should make visible to the linker. As you can see, all of the subroutines have acquired an underscore prefix (as in '_Move'). The Lattice compiler expects this, as do most C compilers.

Next, there are a number of "equates" (EQUR and EQU commands) which function very much like '#define' statements in a C program. They also help to make the logic of the code more readable by eliminating "magic numbers". These equates direct the assembler to replace one symbol with another throughout the listing prior to the assembly process.

Note that registers are treated differently than other expressions: I need EQUR (as opposed to EQU) to establish that the string 'FP' will be used in place of 'A6'. After this we find a "macro definition" of INCR. This functions equivalently to the C '#define INCR' macro in the HOL version of doodit(). After the assembler has performed macro expansion, the statement

```
 INCR   d2
```

will become

```
    addq.l  #OFFS,d2
    cmpi.l  #NODE*OFFS,d2
    blt     .001        * a UNIQUE label is generated each time
    moveq   #0,d2
.001:
```

I hope to explore 68000 macros in a future article.

SETTING UP A LOCAL STACK FRAME

Cotrol is transfered to the label '_doodit' when the C program calls our assembler routine. Figure 1 illustrates this: we allocate a local stack frame and provide ourselves with some memory storage space to play with.

The LINK and UNLK instructions are the primary method for creating local stack frames on the 68000.

The instruction LINK FP,#LVAR accomplishes three things. First, the previous value of FP is pushed on the stack; second, the new value of SP is moved into FP; and third, the value of LVAR (-16) is added to the stack pointer. Figure 3 shows what the stack looks like after the execution of that instruction.

After all this, (a) the new FP points to the location of the previous frame pointer, (b) your local variable space is available at negative offsets to FP (up to -16), (c) any parameters that were pushed on the stack before your routine was called are available at positive offsets to FP, and (d) SP is now a local stack frame, ready for our use.

Make special note that the value supplied to the LINK instruction must be an even, negative value. Register A7 cannot contain odd values. The UNLK instruction, when we get to it, will undo everything LINK has done, in reverse order. Since we are automatically saving and restoring the previous FP, it doesn't matter if the register we use is the same used in the rest of the program. Lattice, in fact, reserves either A5 or A6 as a local frame pointer, depending on a compile-time switch.

We can be sure 'doodit()' won't cause a stomach ache somewhere else in the C code. We do this by saving the current values of all the registers we intend to use, and restoring them later when we're all done. One qualification: on the Amiga, registers A0, A1, D0 and D1 are regarded as system "scratch registers". This means that the calling routine cannot expect their values to be untouched; hence, we don't have to save them.

The 68000 instruction set has a special instruction for circumstances like these: the "MOVE Multiple" or MOVEM command. With it, you can save up to all sixteen registers with a single instruction.

The MOVEM instruction can save registers without predecrement, as in

MOVEM.L D0-D1,LVAR(FP)

You can restore them by reference to the same address 'LVAR(FP)' because they are moved in the same sequence. When you use the instruction with predecrement, for example,

MOVEM.L D2-D7/SP,-(SP)

A7 is the first register moved. When you restore with postincrement, A7 is the last register moved. Think about it! Any other procedure would lead to chaos . . . .

So, before we do anything else, we execute the instruction

MOVEM.L D2-D7/A2,-(SP)

This loads the contents of registers D2 though D7 and A2 into an area right under our local variable space. See Figure 3.

When we finish, before we UNLK, we will execute MOVEM.L

(SP)+,D2-D7/A2 which will restore the registers to the way they were before our subroutine was called.

ARRAYS AND LOOPS IN 68000

Now that it has taken care of its housekeeping chores, the code can turn to the function at hand, that is, executing the '_doodit' algorithm. I'm not going to go into step-by-step details but will simply highlight some of the more interesting features.

Please keep in mind that '_doodit' is more or less a straightforward translation of the C function 'doodit()', and therefore follows the same sequence of operations. To appreciate this isomorphism, you don't really need any understanding of what the algorithm is doing. All you need do is see how each step in one is mirrored in copy-cat step by the other.

Another common way of speeding up assembler versions of HOL routines is in the calculation of array offsets. If you look at a disassembly of what Lattice C does with 'doodit()', you will see a lot of math devoted to array offsets.

MACDOOD.ASM economizes a great deal in this regard by using a special 68000 addressing mode called "address register indirect with displacement".

We can best appreciate it by reference to 'box'. 'box' is a four-element array of 'coord' structures, each of which in turn consists of two long integers, 'x' and 'y'. Since the algorithm spends a lot of time zipping around box[], it is a good subject for optimization.

Figure 3 provides a snapshot of the memory locations containing the elements of the box array. As we have seen, A2 has been loaded with the base address of the array; indeed, A2 can be regarded as pointing either to box or to box[0].x, since they are the same location.

Figure 3: Struct coord box[4] in memory, showing address-register-indirect-with-displacement addressing [D2=2*8=16]

Higher Memory		BOX[3].Y	A2+28
		BOX[3].X	A2+24
	4(A2,D2)	BOX[2].Y	A2+20
	0(A2,D2)	BOX[2].X	A2+16
		BOX[1].Y	A2+12
		BOX[1].X	A2+8
		BOX[0].Y	A2+4
Lower Memory	0(A2)	BOX[0].X	A2+0

On a more primitive microprocessor, it would be necessary to recalculate the value of A2 every time we wanted to address other elements of the array, or other components of their structures.

The 68000 instruction set makes such calculations unnecessary. It allows us to use a separate data register to index the array element, as well as a separate displacement to select out the particular structure component. Suppose we want to move a new value into box[2].y.

First, we make sure that D2 contains the proper array element offset. Since each 'coord' structure takes up eight bytes (2 long words), we put 2*8 = 16 into D2. And because 'y' is the second component in the structure, we need to add in an additional 4 byte displacement.

Given that D2 has been set up properly, we can now move the new value in with a single instruction

```
MOVE.L VALUE,4(A2,D2)
```

If we are at the same time concerned with another element in the same array, we can keep that offset in a different data register. I use D2 to keep the value of 'frst', multiplied by OFFS. OFFS is 8, the size of one 'coord' structure, and multiply that by D3 for 'scnd'. Aren't you glad the Amiga doesn't use an 8088? I am!

We can attain similar economies when it comes to implementing loop structures. Recall that I counseled against making loops external to our assembler routine. Otherwise, we would have to go through the LINK, save, restore, UNLK sequence over and over again. But also we would have been unable to use the 68000's singularly efficient loop-control instructions.

In particular, the DBRA (Decrement and BRAnch) instruction enables an elegant, single instruction analogue of the C loop

```
for (x = SIDE-1; x >= 0;--x)
```

in assembler:

```
    moveq  #SIDE-1,d0    * d0 = x = SIDE-1
XBEG:   .
        .
        .
    dbra   d0,XBEG     * --x; branch unless x < 0     .
```

Each time the DBRA instruction is executed, it decrements the associated data register. If its value has not yet reached -1, a branch is made to the label. I used such loops hoping that the Lattice compiler would utilize DBRA, but alas it didn't.

DBRA is just one member of a whole slew of 'DBcc' instructions with various condition codes: DBEQ, DBNE, DBMI, DBPL, DBGT, DBLT and more, all the way to DBT (Decrement and Branch if True) and DBF (Decrement and Branch if False).

DBRA, in fact, is identical to DBF, since the condition being tested is whether the asssociated data register contains -1 (hexadecimal $FFFFFFFF).

CALLING OTHER LSF ROUTINES

The last feature of '_doodit' we will examine in any detail is the way it calls other local stack frame subroutines. The structure outlined in Figure 2 is our guide.

Usually, we will need to pass some parameters to the routine we are calling, which have to be pushed on the stack. Lattice differs from some other HOL implementations in that it expects them to be pushed on in the reverse of their order in the parameter list of the called routine. For '_boxinit(port, color, box)' we push box 'first', then 'color', finally 'port'.

This leaves them ordered correctly, from low to high memory, as far as the called routine is concerned. Note that one of the advantages of using a frame pointer is that the offsets for addressing the parameters which were passed to us are NOT affected by stack manipulations. The offsets would change if we were using offsets to SP.

We also have to make sure that we push the right kind of values on the stack. Sometimes a routine wants a pointer, that is, the address of a value, and sometimes it wants the value itself.

Finally, we want to make sure that the AMOUNT of space we allocate on the stack for passing the parameters is correct. Lattice C simplifies things here by using long words to pass everything but floating point values. This is another reason why I avoided floats in 'doodit()'. Other HOL's, of course, will have different practices.

After we have appropriately loaded-up the stack, we have to save register D0-D1/A0-A1 if we don't want their contents corrupted. I personally find this requirement on the Amiga a bit irritating. It makes much more sense to me for each routine to be responsible for restoring the contents of all the registers it uses. But I wasn't invited to join the Amiga software design team.

Anyway, this is where '_doodit' finally makes use of the 16 bytes of local variable space it reserved itself with the LINK instruction. Not all of that space is actually used. I gave myself extra, just in case I might need it.

Now at last, we can call the subroutine in question. This automatically pushes the return address on the stack, so that the called routine finds SP pointing to the stack, and the pushed parameters at SP offsets starting at 4. Now the calling routine will LINK, save, restore, UNLK, just like our routine. Indeed, if we were making a recursive call, it would be our routine.

All C routines are functions, which means there can be a return value as likely as not. New values may also have been implicitly "returned", insofar as we have sent pointers, as opposed to values, as parameters. That is, since the called routine has the address of the variable in question, any changes it makes in it will be felt by the calling routine.

Lattice C uses registers D0 and D1 for holding such return values, which of course ties in with their status as scratch registers. A common alternative convention is to allocate space on the stack for the return value. However, in Lattice C at least, if there is a return value, as there is with '_boxinit', (it sends back a new value for color), we have to save it off before we restore any values to D0-D1/A0-A1.

Finally, but certainly not least and not necessarily last, we have to restore the stack to the value it had before we starting

pushing parameters in preparation for the JSR. If we are negligent in this duty, then when we attempt to restore registers in concluding our routine, we will load absolute garbage into them. Bingo, blinking "GURU" ....

Once again, the 68000 instruction set supplies us with a clever command which is often the most efficient means of cleaning up the stack: the "Load Effective Address" (LEA) instruction. It calculates the address of an operand in the ordinary fashion, but then uses that ADDRESS instead of what's in it. Hence, the instruction 'LEA 12(SP),SP' has the effect of adding 12 to the value in the stack pointer, thereby popping off three long words.

TRAILING CLOUDS OF GLORY

But enough is enough. I leave to you the pleasure of uncovering all the other clever and not-so-clever nuances of MACDOOD.ASM.

I WILL be disappointed, however, if I haven't encouraged others to concoct their own assembly language replacements for C routines. It's a lot of fun, and does carry some rewards. For what it's worth, '_doodit' in the assembler version is 40 percent smaller than 'doodit()' in C, and the new DOODLE is approximately 37 percent faster.

Now if I could just get at those Amiga Move and Draw subroutines...

68000 ASSEMBLY LANGUAGE BIBLIOGRAPHY

I relied on the following books when working out the details of the interfacing process described above. You may find others more useful, but I want at least to give credit where it's due. Any idiocy, of course, is mine, all mine.

Gerry Kane, Doug Hawkins, and Lance Leventhal. 68000 ASSEMBLY LANGUAGE PROGRAMMING; McGraw-Hill (1981): approx. 400 pp. This huge book was one of the first out, and it shows. But I learned a lot from it, and continue to lean on it.

Lattice C Compiler Manual; Commodore-Amiga & Lattice (1985): approx. 150 pp. Lattice has been in the C compiler business for some time, and even though their current Amiga offering could use a lot of improvement, their expertise shows through in this manual. Section 5: "68000 Code Generation" is especially relevant to my topic.

M68000 16/32-bit icroprocessor Programmer's Reference Manual, fourth edition, Motorola/Prentice-Hall (1984): 218 pp. This is an absolutely indispensible reference for doing 68000 code. Don't leave home without it.

Leo Scanlon. The 68000: Principles and Programming; Howard Sams (1981): 237 pp. Although much slimmer than the Osborne volume, I often find clearer examples and explanations in here. My explanation of LINK, for instance, was cribbed from Scanlon.

Listing 1 'doodle.def'

```
/*************************************************
**              LISTING 1:  DOODLE.DEF
*************************************************/

#include <exec/types.h>
#include <exec/tasks.h>
#include <exec/libraries.h>
#include <exec/devices.h>
#include <devices/keymap.h>
#include <graphics/copper.h>
#include <graphics/display.h>
#include <graphics/gfxbase.h>
#include <graphics/text.h>
#include <graphics/view.h>
#include <graphics/gels.h>
#include <graphics/regions.h>
#include <devices/keymap.h>
#include <hardware/blit.h>
#include <lattice/stdio.h>
#include <lattice/ctype.h>
#include <libraries/dos.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>

#define INTERNAL FALSE      /* excises doodit() */
#define NODE 4  /* number of corners per figure */
#define SIDE 4    /* number of figures per side */
#define INCR(a) ((++a)<(NODE)?(a):(0))
                /* wraparound counter, 0..NODE-1 */
#define PART 8          /* reduce sides by 1/PART */
#define XM 400  /* pixel width of graphics area */
#define YM 160 /* pixel height of graphics area */
#define XO 115      /* x origin of graphics area */
#define YO 25       /* y origin of graphics area */
#define WHITE 1
#define BLACK 2
```

```
struct coord    /* one pair of x, y coordinates */
   {
   int x, y;
   };


Listing 2 'doodle.c'

/*********************************************
 *          LISTING 2:  DOODLE.C
 *********************************************/

#include <doodle.def>
          /* has other includes, defines, etc. */
struct GfxBase *GfxBase;
          /* library base pointers */
struct IntuitionBase *IntuitionBase;
struct Window *pwndo;      /* my program window */
struct NewWindow progwind;
struct IntuiMessage *message;
struct coord box[NODE];
         /* basic graphics figure */
int color;  /* color we draw it with */

void init();     /* open libraries and windows */
int boxinit();
   /* reinit. colors, corners, and maybe quit */
void quit();     /* bottle up and go */
void doodit();
/* the routine we will replace with 68k assembler
*/

/*********************************************
**  MAIN PROGRAM
**  initialize some stuff, then doodle endlessly
*********************************************/

main()
   {
   struct RastPort *port;

   init();
   port = pwndo->RPort;
   color = WHITE;
   SetAPen(port, color);
   RectFill(port, XO, YO, XO+XM, YO+YM);
   color = boxinit(port, color, box);
   doodit(port, color, box);
   }

/*********************************************
**  change color; reset box corners to original
**  coords; maybe quit
*********************************************/

int boxinit(port, color, box)
   struct RastPort *port;
   int color;
   struct coord box[NODE];

   {
   quit();
   if (color == WHITE)
     color = BLACK;
   else color = WHITE;
   SetAPen(port, color);
   box[0].x = XO;
   box[0].y = YO;
   box[1].x = XO+XM/SIDE;
   box[1].y = YO;
   box[2].x = XO+XM/SIDE;
   box[2].y = YO+YM/SIDE;
   box[3].x = XO;
   box[3].y = YO+YM/SIDE;
   return(color);
   }


#if INTERNAL  /* if FALSE, excises doodit from
doodle */

/*********************************************
**  draw some doodles until you quit (in boxinit)
*********************************************/

void doodit(port, color, box)
   struct RastPort *port;
   int color;
   struct coord box[NODE];

   {
   int frst = 0, scnd = 1;
   int nux, nuy, x, y, xl, yl;

   while(TRUE)
      {
      for (x = SIDE-1; x >= 0; --x)
         for (y = SIDE-1; y >= 0; --y)
            {
            xl = XM*x/SIDE;
            yl = YM*y/SIDE;
         Move(port, box[frst].x+xl,box[frst].y+yl);
         Draw(port, box[scnd].x+xl,box[scnd].y+yl);
            }
```

```
        nux = box[frst].x +
              (box[scnd].x - box[frst].x)/PART;
        nuy = box[frst].y +
              (box[scnd].y - box[frst].y)/PART;
        if ((box[frst].x == nux) &&
           (box[frst].y == nuy))
           {
           color = boxinit(port, color, box);
            /* program can exit here */
           frst = 0;
           scnd = 1;
           }
        else
           {
           box[frst].x = nux;
           box[frst].y = nuy;
           frst = INCR(frst);
           scnd = INCR(scnd);
           }
     }
  }

#endif


/***********************************************
**  check for exit signal; if so, clean up and go
***********************************************/

void quit()

  {
  if (message = (struct IntuiMessage
*)GetMsg(pwndo->UserPort))
     if (message->Class == CLOSEWINDOW)
        {
        ReplyMsg(message);
        CloseWindow(pwndo);
        CloseLibrary(GfxBase);
        CloseLibrary(IntuitionBase);
        exit(0);
        }
  }


/***********************************************
**  open appropriate libraries and windows, etc.
***********************************************/

char ilib[] = "intuition.library";
              /* assorted string constants */
char glib[] = "graphics.library";
char fmsg[] = "failed.\n";

void init()

  {
  if ((IntuitionBase = (struct IntuitionBase *)
       OpenLibrary(ilib, 0)) == 0)
       {
     printf("%s %s", ilib, fmsg);
     exit();
     }
  if ((GfxBase = (struct GfxBase *)
     OpenLibrary(glib, 0)) == 0)
     {
     printf("%s %s", glib, fmsg);
     CloseLibrary(IntuitionBase);
     exit();
     }
  progwind.LeftEdge      = 0;
  progwind.TopEdge       = 0;
  progwind.Width         = 640;
  progwind.Height        = 200;
  progwind.DetailPen     = 0;
  progwind.BlockPen      = 1;
  progwind.Flags         =
        ACTIVATE | SIMPLE_REFRESH | WINDOWCLOSE;
  progwind.IDCMPFlags    = CLOSEWINDOW;
  progwind.FirstGadget   = NULL;
  progwind.CheckMark     = NULL;
  progwind.Title         =
                "Jim Dandy Doodle All The Day";
  progwind.Screen        = NULL;
  progwind.BitMap        = NULL;
  progwind.MinWidth      = 640;
  progwind.MinHeight     = 200;
  progwind.MaxWidth      = 640;
  progwind.MaxHeight     = 200;
  progwind.Type          = WBENCHSCREEN;
  if ((pwndo = (struct Window *)
        OpenWindow(&progwind)) == 0)
     {
     printf("program window %s", fmsg);
     CloseLibrary(GfxBase);
     CloseLibrary(IntuitionBase);
     exit();
     }
  }
```

Listing 3 'macdood.asm'

```
*************************************************
**        LISTING 3:  MACDOOD.ASM
*************************************************
**
**    register usage (see C version of doodit()):
**
**       a2 -> box (= box[0].x)
**       a6 =  frame pointer (FP)
**       a7 =  stack pointer (SP)
**       d0 =  x
**       d1 =  y
**       d2 =  frst
**  (*8 for indexing "coords,"
**  i.e. double long words)
**       d3 =  scnd (ditto)
**       d4 =  xl; nux
**       d5 =  yl; nuy
**       d6 =  box[frst/scnd].x + xl
**       d7 =  box[frst/scnd].y + yl
**    let the linker know what we need, and what we
**    got here

      xref  _Move,_Draw,_boxinit
      xdef  _doodit


**    equates

FP    equr  a6    * a6 is the local frame pointer
SP    equr  a7    * a7 is the stack pointer

LVAR  equ   -16  * allocate 16 bytes for loc. var.
NODE  equ   4  * number of corners in basic figure
SIDE  equ   4    * number of figures per side
PART  equ   8  * sides reduced by 1/PART each time
XM    equ   400  * pixel width of graphics area
YM    equ   160  * pixel height of graphics area
OFFS  equ   8    * index amt = 8 bytes per coord

**    68000 macro version of '#define INCR(a)
**    ((++a)<(NODE)?(a):(0))'

INCR  macro
      addq.l   #OFFS,\1
      cmpi.l   #NODE*OFFS,\1
      blt      \@
      moveq    #0,\1
\@:
      endm

**    execution begins here

_doodit:
      link     FP,#LVAR  * alloc local stack frame
      movem.l  d2-d7/a2,-(SP) * save registers
      move.l   16(FP),a2      * a2 = address of box

STR1: moveq    #0,d2          * frst = 0
      moveq    #OFFS,d3       * scnd = 1 (*8)

STR2: moveq    #SIDE-1,d0     * x = SIDE-1

XBEG: moveq    #SIDE-1,d1     * y = SIDE-1

YBEG: move.l   d0,d4          * xl = x
      mulu     #XM,d4         * xl = x*xm
      divu     #SIDE,d4       * xl = (x*xm)/SIDE
      move.l   d1,d5          * yl = y
      mulu     #YM,d5         * yl = y*ym
      divu     #SIDE,d5       * yl = (y*ym)/SIDE
      move.l   0(a2,d2),d6    * d6 = box[frst].x
      add.w    d4,d6          * d6 = box[frst].x+xl
      move.l   4(a2,d2),d7    * d7 = box[frst].y
      add.w    d5,d7          * d7 = box[frst].y+yl
      move.l   d7,-(SP)
        * push box[frst].y+yl on stack
      move.l   d6,-(SP)
        * push box[frst].x+xl on stack
      move.l   8(FP),-(SP)    * push port on stack
      movem.l  d0-d1,LVAR(FP)
        * keep d0 & d1 from mischief

      jsr      _Move
        * move cursor (Amiga ROM module)
      move.l   0(a2,d3),d6    * d6 = box[scnd].x
      add.l    d4,d6          * d6 = box[scnd].x+xl
      move.l   4(a2,d3),d7    * d7 = box[scnd].y
      add.l    d5,d7          * d7 = box[scnd].y+yl
      move.l   d6,4(SP)
        * move box[scnd].x+xl into stack
      move.l   d7,8(SP)
        * move box[scnd].y+yl into stack

      jsr      _Draw
        * draw line (Amiga ROM module)

      movem.l  LVAR(FP),d0-d1
        * restore d0 & d1 (finally)
      lea      12(SP),SP
        * clean up stack (finally)
```

```
        dbra    d1,YBEG  * --y; branch unless y < 0

        dbra    d0,XBEG  * --x; branch unless x < 0
        move.l  0(a2,d3),d4  * nux = box[scnd].x
        sub.l   0(a2,d2),d4
                  * nux = nux - box[frst].x
        divs    #PART,d4     * nux = nux/PART
        ext.l   d4
                  * (dump remainder in upper word)
        add.l   0(a2,d2),d4
                  * nux = nux + box[frst].x
        move.l  4(a2,d3),d5  * nuy = box[scnd].y
        sub.l   4(a2,d2),d5
                  * nuy = nuy - box[frst].y
        divs    #PART,d5     * nuy = nuy/PART
        ext.l   d5
                  * (dump remainder in upper word)
        add.l   4(a2,d2),d5
                  * nuy = nuy + box[frst].y
        cmp.l   0(a2,d2),d4  * box[frst].x = nux?
        bne     MORE         *    if not, branch

        cmp.l   4(a2,d2),d5  * box[frst].y = nuy?
        bne     MORE         *    if not, branch

        move.l  16(FP),-(SP) * push box on stack
        move.l  12(FP),-(SP) * push color on stack
        move.l  8(FP),-(SP)  * push port on stack

        jsr     _boxinit
                * reinit. box corners, perhaps quit
        lea     12(SP),SP    * clean up stack
        move.l  d0,12(FP)
                  * get return value = color
        beq     FINI
                  * color = 0 means we're done
        bra     STR1
                  * branch (reinit. frst & scnd)

MORE:   move.l  d4,0(a2,d2)    * box[frst].x = nux
        move.l  d5,4(a2,d2)    * box[frst].y = nuy
        INCR    d2             * frst = INCR(frst)
        INCR    d3             * scnd = INCR(scnd)
        bra     STR2           * do it all over again


FINI    movem.l (SP)+,d2-d7/a2
                  * restore all used registers
        unlk    FP   * deallocate local stack frame
        rts                  * return
        end                  * that's all, folks!
```

•AC•

AMICUS and Fred Fish Public Domain Software Library

This software is collected from user groups and electronic bulletin boards around the nation. Each disk is nearly full, and is fully accessible from the Workbench. If source code is provided for any program, then the executable version is also present. This means that you don't need the C compiler to run these programs. An exception is granted for those programs only of use to people who own a C compiler.

Note: Each description line below may include something like 'S-O-E-D', which stands for 'source, object file, executable and documentation'. Any combination of these letters indicates what forms of the program are present. Basic programs are presented entirely in source code format.

AMICUS Disk 1

ABasic programs: Graphics

3DSolids	3d solids modeling program w/sample data files
Blocks	draws blocks
Cubes	draws cubes
Durer	draws pictures in the style of Durer
FScape	draws fractal landscapes
Hidden	3D drawing program, w/ hidden line removal
JPad	simple paint program
Optical	draw several optical illusions
PaintBox	simple paint program
Shuttle	draws the Shuttle in 3d wireframe
SpaceArt	graphics demo
Speaker	speech utility
Sphere	draws spheres
Spiral	draws color spirals
ThreeDee	3d function plots
Topography	artificial topography
Wheels	draws circle graphics
Xenos	draws fractal planet landscapes

ABasic programs: Tools

AddressBook	simple database program for addresses
CardFile	simple card file database program
Demo	multiwindow demo
KeyCodes	shows keycodes for a key you press
Menu	run many ABasic programs from a menu
MoreColors	way to get more colors on the screen at once, using aliasing
shapes	simple color shape designer SpeakIt speech and narrator demo

ABasic programs: Games

BrickOut	classic computer brick wall game
Othello	also known as 'go'
Saucer	simple shoot-em-up game
Spelling	simple talking spelling game
ToyBox	selectable graphics demo

ABasic programs: Sounds

Entertainer	plays that tune
HAL9000	pretends it's a real computer
Police	simple police siren sound
SugarPlum	plays "The Dance of the Sugarplum Fairies"

C programs:

ATerm	simple terminal program, S-E
cc	aid to compiling with Lattice C
decvnt	opposite of CONVERT for cross developers
Dotty	source code to the 'dotty window demo
echox	unix-style filename expansion, partial S, O-D
fasterfp	explains use of fast-floating point math
FixDate	fixes future dates on all files on a disk, S-E
freedraw	simple Workbench drawing program,S-E
GfxMem	graphic memory usage indicator, S-E
Grep	searches for a given string in a file, with documentation
ham	shows off the hold-and-modify method of color generation
IBM2Amiga	fast parallel cable transfers between an IBM and an Amiga
Mandel	Mandelbrot set program, S-E
moire	patterned graphic demo, S-E
objfix	makes Lattice C object file symbols visible to Wack, S-E
quick	quick sort strings routine
raw	example sample window I/O
setlace	turns on interlace mode, S-E
sparks	qix-type graphic demo, S-E

Other executable programs:

SpeechToy	speech demonstration
WhichFont	displays all available fonts

Texts:

68020	describes 68020 speedup board from CSA
Aliases	explains uses of the ASSIGN command
Bugs	known bug list in Lattice C 3.02
CLICard	reference card for AmigaDOS CLI
CLICommands	guide to using the CLI
Commands	shorter guide to AmigaDOS CLI commands
EdCommands	guide to the ED editor
Filenames	AmigaDOS filename wildcard conventions
HalfBright	explains rare graphics chips that can do more colors
ModemPins	description of the serial port pinout
RAMdisks	tips on setting up your RAM: disk
ROMWack	tips on using ROMWack
Sounds	explanation of the Instrument demo sound file format
Speed	refutation of the Amiga's CPU and custom chip speed
WackCmds	tips on using Wack

AMICUS Disk 2

C programs:

alib	AmigaDOS object library manager , S-E
ar	text file archive program, S-E
fixobj	auto-chops executable files
shell	simple CLI shell, S-E
sq, usq	file compression programs, S-E
YachtC	a familiar game, S-E
Make	a simple 'make' programming utility, S-E
Emacs	an early version of the Amiga text editor, S-E-D

Assembler programs:

bsearch.asm	binary search code
qsort.asm	Unix compatible qsort() function, source and C test program
setjmp.asm	setjmp() code for Lattice 3.02
SVprintf	Unix system V compatible printf()
trees.o	Unix compatible tree() function, O-D

(This disk formerly had IFF specification files and examples. Since this spec is constantly updated, the IFF spec files have been moved to their own disk in the AMICUS collection. They are not here.)

John Draper Amiga Tutorials:

Animate	describes animation algorithms
Gadgets	tutorial on gadgets
Menus	learn about Intuition menus

AMICUS Disk 3

C programs:

Xref	a C cross-reference gen., S-E
6bitcolor	extra-half-bright chip gfx demo, S-E
Chop	truncate (chop) files down to size, S-E
Cleanup	removes strange characters from text files
CR2LF	converts carriage returns to line feeds in Amiga files, S-E
Error	adds compile errors to a C file, S
Hello	window ex. from the RKM, S
Kermit	generic Kermit implementation, flakey, no terminal mode, S-E
Scales	sound demo plays scales, S-E
Skew8	Rubik cube demo in hi-res colors, S-E

AmigaBasicProgs(dir)

Automata	cellular automata simulation
CrazyEights	card game
Graph	function graphing programs
WitchingHour	a game

ABasiC programs:

Casino	games of poker, blackjack, dice, and craps
Gomoku	also known as 'othello'
Sabotage	sort of an adventure game

Executable programs:

Disassem	a 68000 disassembler, E-D
DpSlide	shows a given set of IFF pictures, E-D
Arrange	a text formatting program, E-D

Assembler programs:

Argoterm	a terminal program with speech and Xmodem, S-E

AMICUS Disk 4 **Files from the original Amiga Technical BBS**

Note that some of these files are old, and refer to older versions of the operating system. These files came from the Sun system that served as Amiga technical support HQ for most of 1985. These files do hot carry a warranty, and are for educational purposes only. Of course, that's not to say they don't work.

Complete and nearly up-to-date C source to 'image.ed', an early version of the Icon Editor. This is a little flaky, but compiles and runs.

An Intuition demo, in full C source, including files: demomenu.c, demomenu2.c, demoreq.c, getascii.c, idemo.c, idemo.guide, idemo.make, idemoall.h, nodos.c, and txwrite.c

addmem.c	add external memory to the system
bobtest.c	example of BOB use
consoleIO.c	console IO example
creaport.c	create and delete ports
creastdi.c	create standard I/O requests
creatask.c	creating task examples
diskio.c	example of track read and write
dotty.c	source to the 'dotty window' demo
dualplay.c	dual playfield example
flood.c	flood fill example
freemap.c	old version of 'freemap'
geltools.c	tools for VSprites and BOBs
gfxmem.c	graphic memory usage indicator
hello.c	window example from RKM
inputdev.c	adding an input handler to the input stream

joystik.c	reading the joystick
keybd.c	direct keyboard reading
layertes.c	layers examples
mousport.c	test mouse port
ownlib.c,	
ownlib.asm	example of making your own library with Lattice
paratest.c	tests parallel port commands
seritest.c	tests serial port commands
serisamp.c	example of serial port use
prinintr.c	sample printer interface code
prtbase.h	printer device definitions
regintes.c	region test program
setlace.c	source to interlace on/off program
setparallel.c	set the attributes of the parallel port
SetSerial.c	set the attributes (parity, data bits) of the serial port
singplay.c	single playfield example
speechtoy.c	source to narrator and phonetics demo
timedely.c	simple timer demo
timer.c	exec support timer functions
timrstuf.c	more exec support timer functions
WhichFont.c	loads and displays all available system fonts

process.i and prtbase.i assmebler include files:

autorqstr.txt	warnings of deadlocks with autorequesters
consoleIO.txt	copy of the RKM console I/O chapter
diskfont.txt	warning of disk font loading bug
fullfunc.txt	list of #defines, macros, functions
inputdev.txt	preliminary copy of the input device chapter

License information on Workbench distribution license printer pre-release copy of the chapter on printer drivers, from RKM 1.1 v11fd.txt 'diff' of .fd file changes from version 1.0 to 1.1 v28v1.diff 'diff' of include file changes from version 28 to 1.0

AMICUS Disk 5 Files from the Amiga Link / Amiga Information Network

Note that some of these files are old, and refer to older versions of the operating system. These files are from Amiga Link. For a time, Commodore supported Amiga Link, aka AIN, for online developer technical support. It was only up and running for several weeks. These files do not carry a warranty, and are for educational purposes only. Of course, that's not to say they don't work.

A demo of Intuition menus called 'menudemo', in C source

whereis.c	find a file searching all subdirectories
bobtest.c	BOB programming example
sweep.c	sound synthesis example

Assembler files:

mydev.asm	sample device driver
mylib.asm	sample library example
mylib.i	
mydev.i	
asmsupp.i	

macros.i assembler include files:

Texts:

amigatricks	tips on CLI commands
extdisk	external disk specification
gameport	game port spec
parallel	parallel port spec
serial	serial port spec
v1.1update	list of new features in version 1.1
v1.1h.txt	'diff' of include file changes from version 1.0 to 1.1

Files for building your own printer drivers, including dospecial.c, epsondata.c, init.asm, printer.c, printer.link, printertag.asm, render.c, and wait.asm. This disk does contain a number of files describing the IFF specification. These are not the latest and greatest files, but remain here for historical purposes. They include text files and C source examples. The latest IFF spec is elsewhere in this library.

AMICUS Disk 6 IFF Pictures

This disk includes the DPSlide program, which can view a given series of IFF pictures, and the 'showpic' program, which can view each file at the click of an icon, and the 'saveilbm' program, to turn any screen into an IFF picture. The pictures include a screen from ArticFox, a Degas dancer, the guys at Electronic Arts, a gorilla, horses, King Tut, a lighthouse, a screen from Marble Madness, the Bugs Bunny Martian, a still from an old movie, the Dire Straits moving company, a screen from Pinball Contruction Set, a TV newcaster, the PaintCan, a world map, a Porsche, a shuttle mission patch, a tyrannosaurus rex, a planet view, a VISA card, and a ten-speed.

AMICUS Disk 7 DigiView HAM demo picture disk

This disk has pictures from the DigiView hold-and-modify video digitizer. It includes the ladies with pencils and lollypops, the young girl, the bulldozer, the horse and buggy, the Byte cover, the dictionary page, the robot and Robert. This includes a program to view each picture separately, and all together as separate, slidable screens.

AMICUS Disk 8

C programs:

Browse	view text files on a disk, using menus S-E-D
Crunch	removes comments and white space from C files, S-E
IconExec	EXECUTE a series of commands from Workbench S-E
PDScreen	
Dump	dumps Rastport of highest screen to printer
SetAlternate	sets a second image for an icon, when clicked once S-E
SetWindow	makes windows for a CLI program to run under Workbench S-E
SmallClock	a small digital clock that sits in a window menu bar
Scrimper	the screen printer in the fourth Amazing Computing, S-E

Amiga Basic Programs:

(**Note:** Many of these programs are present on AMICUS Disk 1. Several of these were converted to Amiga Basic, and are included here.)

AddressBook	a simple address book database
Ball	draws a ball
Cload to	program to convert Compuserve hex files binary, S-D
Clue	the game, Intuition driven
ColorArt	art drawing program
DeluxeDraw	the drawing program in the 3rd issue of Amazing Computing, S-D
Eliza	conversational computer psychologist
Othello	the game, as known as 'go'
RatMaze	3D ratmaze game
ROR	boggling graphics demo
Shuttle	draws 3D pictures of the space shuttle
Spelling	simple spelling program
YoYo yo	wierd zero-gravity yo-yo demo, tracks yo-to the mouse

Executable programs:

3Dcube	Modula-2 demo of a rotating cube
AltIcon	sets a second icon image, displayed when the icon is clicked
AmigaSpell	a slow but simple spelling checker, E-D
arc	the ARC file compression program, must-have for telecom, E-D
Bertrand	graphics demo
disksalvage	a program to rescue trashed disks, E-D
KwikCopy	a quick but nasty disk copy program: ignores errors, E-D
LibDir	lists hunks in an object file E-D
SaveILBM	saves any screen as an IFF picture E-D ??
ScreenDump	shareware screen dump program, E only
StarTerm	version 2.0, term program, Xmodem E-D

Texts:

LatticeMain	tips on fixing _main.c in Lattice
GDiskDrive	make your own 5 1/4 drive
GuruMed	explains the Guru numbers
Lat3.03bugs	bug list of Lattice C version 3.03
MForgeRev	user's view of the MicroForge hard drive
PrintSpooler	EXECUTE-based print spooling program

.BMAP files:

These are the necessary links between Amiga Basic and the system libraries. To take advantage of the Amiga's capabilities in Basic, you need these files. BMAPs are included for 'clist', 'console', 'diskfont', 'exec', 'icon', 'intuition', 'layers', 'mathffp', mathieeedoubas', 'mathieeesingbas', 'mathtrans', 'potgo', 'timer' and 'translator'.

AMICUS Disk 9

Amiga Basic Programs:

FlightSim	simple flight simulator program
HuePalette	explains Hue, Saturation, and Intensity
Requester	ex. of doing requesters from Amiga Basic
ScrollDemo	demonstrates scrolling capabilities
Synthesizer	sound program
WorldMap	draws a map of the world

Executable programs:

Boing!	latest Boing! demo,with selectable speed, E
Brush2C	converts an IFF brush to C data instructions, initialization code, E
Brush2Icon	converts IFF brush to an icon, E
Dazzle	graphics demo, tracks to mouse, E
DeciGEL	assembler program for stopping 68010 errors, S-E-D
Klock	menu-bar clock and date display, E
life	the game of life, E
TimeSet	Intuition-based way to set the time and date, E
MEmacs	another Emacs, more oriented to word processing, S-E-D
MyCLI	a CLI shell, works without the Workbench, S-E-D

Texts:

FnctnKeys	explains how to read function keys from Amiga Basic
HackerSln	explains how to win the game 'hacker'
Ist68010	guide to installing a 68010 in your Amiga
PrinterTip	tips on sending escape sequences to your printer
StartupTip	tips on setting up your startup-sequence file
XfrmrReview	list of programs that work with the Transformer

Printer Drivers:

Printer drivers for the Canon PJ-1080A, the C Itoh Prowriter, an improved Epson driver that eliminates streaking, the Epson LQ-800, the Gemini Star-10, the NEC 8025A, the Okidata ML-92, the Panasonic KX-P10xx family, and the Smith-Corona D300, with a document describing the installation process.

AMICUS Disk 10 Instrument sound demos

This is an icon-driven demo, circulated to many dealers. It includes the sounds of an acoustic guitar, an alarm, a banjo, a bass guitar, a boink, a calliope, a car horn, claves, water drip, electric guitar, a flute, a harp arpegio, a kickdrum, a marimba, a organ minor chord, people talking, pigs, a pipe organ, a Rhodes piano, a saxophone, a sitar, a snare drum, a steel drum, bells, a vibrophone, a violin, a wailing guitar, a horse whinny, and a whistle.

Fred Fish Disk 1:

amigademo	Graphical benchmark for comparing amigas.
amigaterm	simple communications program with Xmodem
balls	simulation of the "kinetic thingy" with balls on strings
colorful	Shows off use of hold-and-modify mode.
dhrystone	Dhrystone benchmark program.
dotty	Source to the "dotty window" demo on the Workbench disk.
freedraw	A small "paint" type program with lines, boxes, etc.
gad	John Draper's Gadget tutorial program
gfxmem	Graphical memory usage display program
halfbrite	demonstrates "Extra-Half-Brite" mode, if you have it
hello	simple window demo
lattffp	accessing the Motorola Fast Floating Point l ibrary from C
palette	Sample program for designing color palettes.
trackdisk	Demonstrates use of the trackdisk driver.
requesters	John Draper's requester tutorial and example program.
speech	Sample speech demo program. Stripped down "speechtoy".
speechtoy	Another speech demo program.

Fred Fish Disk 2:

alib	Object module librarian.
cc	Unix-like frontend for Lattice C compiler.
dbug	Macro based C debugging package. Machine independent.
make	Subset of Unix make command.
make2	Another make subset command.
microemacs	Small version of emacs editor, with macros, no extensions
portar	Portable file archiver.
xrf	DECUS C cross reference utility.

Fred Fish Disk 3:

gothic	Gothic font banner printer.
roff	A "roff" type text formatter.
ff	A very fast text formatter
cforth	A highly portable forth implementation. Lots of goodies.
xlisp	Xlisp 1.4, not working correctly.

Fred Fish Disk 4:

banner	Prints horizontal banner
bgrep	A Boyer-Moore grep-like utility
bison	CNU Unix replacement 'yacc', not working.
bm	Another Boyer-Moore grep-like utility
grep	DECUS grep
kermit	simple portable Kermit with no connect mode.
MyCLI	Replacement CLI for the Amiga. Version 1.0
mandel	A Mandelbrot set program, by Robert French and RJ Mical

Fred Fish Disk 5:

cons	Console device demo program with supporting macro routines.
freemap	Creates a visual diagram of free memory
input.dev	sample input handler, traps key or mouse events
joystick	Shows how to set up the gameport device as a joystick.
keyboard	demonstrates direct communications with the keyboard.
layers	Shows use of the layers library
mandelbrot	IFF Mandelbrot program
mouse	hooks up mouse to right joystick port
one.window	console window demo
parallel	Demonstrates access to the parallel port.
printer	opening and using the printer, does a screen dump, not working
print.support	Printer support routines, not working.
proctest	sample process creation code, not working
region	demos split drawing regions
samplefont	sample font with info on creating your own
serial	Demos the serial port
singlePlayfield	Creates 320 x 200 playfield
speechtoy	latest version of cute speech demo
speech.demo	simplified version of speechtoy, with IO requests
text.demo	displays available fonts
timer	demos timer.device use
trackdisk	demos trakcdisk driver

Fred Fish Disk 6:

compress	like Unix compress, a file squeezer
dadc	analog clock impersonator
microemacs	upgraded version of microemacs from disk 2
mult	removes multiple occuring lines in files
scales	demos using sound and audio functions
setparallel	Allows changing parallel port parameters
setserial	Allows changing serial port parameters.
sortc	quicksort based sort program, in C
stripc	Strips comments and extra whitespace from C source

Fred Fish Disk 7:

This disk contains the executables of the game Hack, version 1.0.1.

Fred Fish Disk 8:

This disk contains the C source to Hack on disk 7.

Fred Fish Disk 9:

moire	Draws moire patterns in black and white
MVP-FORTH	Mountain View Press Forth, version 1.00.03A. A shareware version of FORTH from Fantasia Systems.
proff	a more powerful text formatting program
setlace	Program to toggle interlace mode on and off.
skewb	a rubic's cube type demo
sparks	moving snake Graphics demo

Fred Fish Disk 10:

conquest	An interstellar adventure simulation game
dehex	convert a hex file to binary
filezap	Patch program for any type of file.
fixobj	Strip garbage off Xmodem transferred files.
iff	Routines to read and write iff format files.
ld	simple directory program
ls	Minimal UNIX ls, with Unix-style wildcarding, in C
sq,usq	file squeeze and unsqueeze
trek73	Star Trek game
yachtc	Dice game.

Fred Fish Disk 11:

dpslide	slide show program for displaying IFF images with miscellaneous pictures

Fred Fish Disk 12:

amiga3d	Shows a rotating 3 dimensional solid "Amiga sign".
ArgoTerm	a terminal emulator program, written in assembler
arrow3d	Shows a rotating 3 dimensional wire frame arrow.
id4	directory listing program
IconExec	
SetWindow	two programs for launching programs from Workbench that presently only work under CLI.
SetAlternate	Makes an icon show a second image when clicked once
StarTerm	terminal emulator, with ASCII Xmodem, dialer, more.

Fred Fish Disk 13:

A Bundle of Basic programs, including:

Jpad	toybox	ezspeak	mandlebrot
xmodem	3dsolids	addbook	algebra
ror	amgseq1	amiga-copy	band
bounce	box	brickout	canvas
cardfi	circle	colorcircles	Copy
cubes1	cutpaste	date	dogstar
dragon	draw	dynamictriangle	
Eliza	ezterm	fillibuster	fractal
fscape	gomoku	dart	haiku
hal9000	halley	hauntedM	hidden
join	loz	mandel	menu
minipaint	mouse	Orthello	patch
pena	pinwheel	gbox	random-circles
Readme	rgb	rgbtest	Rord
sabotage	salestalk	shades	shapes
shuttle	sketchpad	spaceart	speak
speach	speecheasy	spell	sphere
spiral	striper	superpad	suprshr
talk	terminal	termtest	tom
topography	triangle	wheels	xenos
xmostriper			

(note: some programs are Abasic, most are Amigabasic, and some programs are presented in both languages)

Fred Fish Disk 14:

amiga3d	update of #12, includes C source to a full hidden surface removal and 3D graphics
beep	Source for a function that generates a beep sound
dex	extracts text from within C source files
dimensions	demonstrates N dimensional graphics
filezap	update of disk 10, a file patch utility
gfxmem	update of disk 1, graphic memory usage indicator
gl	converts IFF brush files to Image struct, in C text.
pdterm	simple ANSI VT100 terminal emulator, in 80 x 25 screen
shell	simple Unix 'csh' style shell
termcap	mostly Unix compatible 'termcap' implementation.

Fred Fish Disk 15:

Blobs	graphics demo, like Unix 'worms'
Clock	simple digital clock program for the title bar
Dazzle	An eight-fold symmetry dazzler program. Really pretty!
Fish	double buffered sequence cycle animation of a fish
Monopoly	A really nice monopoly game written in AbasiC.
OkidataDump	Okidata ML92 driver and WorkBench screen dump program.
Polydraw	A drawing program written in AbasiC.
Polyfractals	A fractal program written in AbasiC.

Fred Fish Disk 16:

Complete copy of the latest developer IFF disk

Fred Fish Disk 17:

The NewTek Digi-View video digitizer HAM demo disk

Fred Fish Disk 18:

AmigaDisplay	dumb terminal program with bell, selectable fonts
Ash	Prerelease C Shell-like shell program, history, loops, etc.
Browser	wanders a file tree, displays files, all with the mouse
MC68010	docs on upgrading your Amiga to use a 68010
Multidim	rotate an N dimensional cube with a joystick
PigLatin	SAY command that talks in Pig Latin
Scrimper	Screen image printer
Xlisp1.6	source, docs, and executable for a Lisp interpreter.

Fred Fish Disk 19:

BlackJack	text-oriented blackjack game
JayMinerSlides	Slides by Jay Miner, Amiga graphics chip designer, showing flowchart of the Amiga internals, in 640 x 400.
Keymap_Test	test program to test the keymapping routines
LockMon	Find unclosed file locks, for programs that don't clean u**Fred Fish Disk 20:**
AmigaToAtari	converts Amiga object code to Atari format
DiskSalv	program to recover files from a trashed AmigaDOS disk.
Hash	example of the AmigaDOS disk hashing function
Hd	Hex dump utility ala Computer Language magazine, April 86
MandelBrots	Mandelbrot contest winners
MultiTasking	Tutorial and examples for Exec level multitasking
Pack	strips whitespace from C source
PortHandler	sample Port-Handler program that performs. Shows BCPL environment clues.
Random	Random number generator in assembly, for C or assembler.
SetMouse2	sets mouse port to right or left port.
SpeechTerm	terminal emulator with speech capabilities, Xmodem
TxEd	Demo editor from Microsmiths Charlie Heath

Fred Fish Disk 21

This is a copy of Thomas Wilcox's Mandelbrot Set Explorer disk. Very good!

Fred Fish Disk 22

This disk contains two new "strains" of microemacs.

Lemacs	version 3.6 by Daniel Lawrence. For Unix V7, BSD 4.2, Amiga, MS-DOS, VMS. Uses Amiga function keys, status line, execute, startup files, more.
Pemacs	By Andy Poggio. New features include <ALT> keys as Meta keys, mouse support, higher priority, backup files, word wrap, function keys.

Fred Fish Disk 23

Disk of source for MicroEmacs, several versions for most popular operating systems on micros and mainframes. For people who want to port MicroEmacs to their favorite machine.

Fred Fish Disk 24:

Conques	interstaller adventure simulation game
Csh	update to shell on Disk 14, with built in commands,named variables, substitution.
Modula-2	A pre-release version of the single pass Modula-2 compiler originally developed for MacIntosh at ETHZ. This code was transmitted to the AMIGA and is executed on the AMIGA using a special loader. Binary only.

Fred Fish Disk 25

Graphic Hack graphic version of the game on disks7&8

Fred Fish Disk 26

UnHun	Processes the Amiga "hunk" loadfiles. Collect code, data, and bss hunks together, allows individual specification of code, data, and bss origins, and generates binary file with format reminiscent of Unix "a.out"format. The output file can be easily processed by a separate program to produce Motorola "S-

records" suitable for downloading to PROM programmer. By Eric Black.

C-kermit	Port of the Kermit file transfer program and server.
Ps	Display and set process priorities
Archx	Yet another program for bundling up text files and mailing or posting them as a single file unit.

Fred Fish Disk 27

ABdemos	AmigaBasic demos fromCarolynScheppner.
NewConvertFD	creates .bmaps from fd files.
BitPlanes	finds addresses of and writes to bitplanes of the screen's bitmap.
AboutBmaps	is a tutorial on creation and use of bmaps.
LoadILBM	loads and displays IFF ILBM pics.
LoadACBM	loads and displays ACBM pics.
ScreenPrint	creates a demo screen and dumps it to a graphic printer.
Disassem	Simple 68000 disassembler. Reads standard Amiga object files and disassembles the code sections. Data sections are dumped in hex. The actual disassember routines are set up to be callable from a user program so instructions in memory can be disassembled dynamically. By Bill Rogers.
DvorakKeymap	Example of a keymap structure for the Dvorak keyboard layout. Untested but included because assembly examples are few and far between.By Robert Burns of C-A.
Hypocycloids	Spirograph, from Feb. 84 Byte.
LinesDemo	Example of proportional gadgets to scroll a SuperBitMap.
MemExpansion	Schematics and directions for building your own homebrew 1 Mb memory expansion, by Michael Fellinger.
SafeMalloc	Program to debug 'malloc()' calls
ScienceDemos	Convert Julian to solar and sidereal time, stellar positions and radial velocity epoch calculations and Galilean satellite plotter. By David Eagle.

Fred Fish Disk 28

Basic games	by David Addison: Backgammon, Cribbage, Milestone, Othello
Cpp	DECUS 'cpp' C preprocessor, and a modified 'cc' that knows about the 'cpp', for Manx C.
Shar	Unix-compatible shell archiver, for packing files for travel.
SuperBitMap	Example of using a ScrollLayer, syncing SuperBitMaps for printing, and creating dummy RastPorts.

Fred Fish Disk 29

AegisDraw Demo	Demo without save and no docs.
Animator Demo	Player for Aegis Animator filesUnix-like ront-end for Manx C.
Enough	Tests for existance of system resources, files, devices.
Rubik	Animated Rubik's cube program
StringLib	Public domain Unix string library functions.
Vt100	VT-100 terminal emulator with Kermit and Xmodem protocols

Fred Fish Disk 30

Several shareware programs. The authors request a donation if you find their program useful, so they can write more software.

BBS	an Amiga Basic BBS by Ewan Grantham
FineArt	Amiga art
FontEditor	edit fonts, by Tim Robinson
MenuEditor	Create menus, save them as C source, by David Pehrson
StarTerm3.0	Very nice telecommunications by Jim Nangano

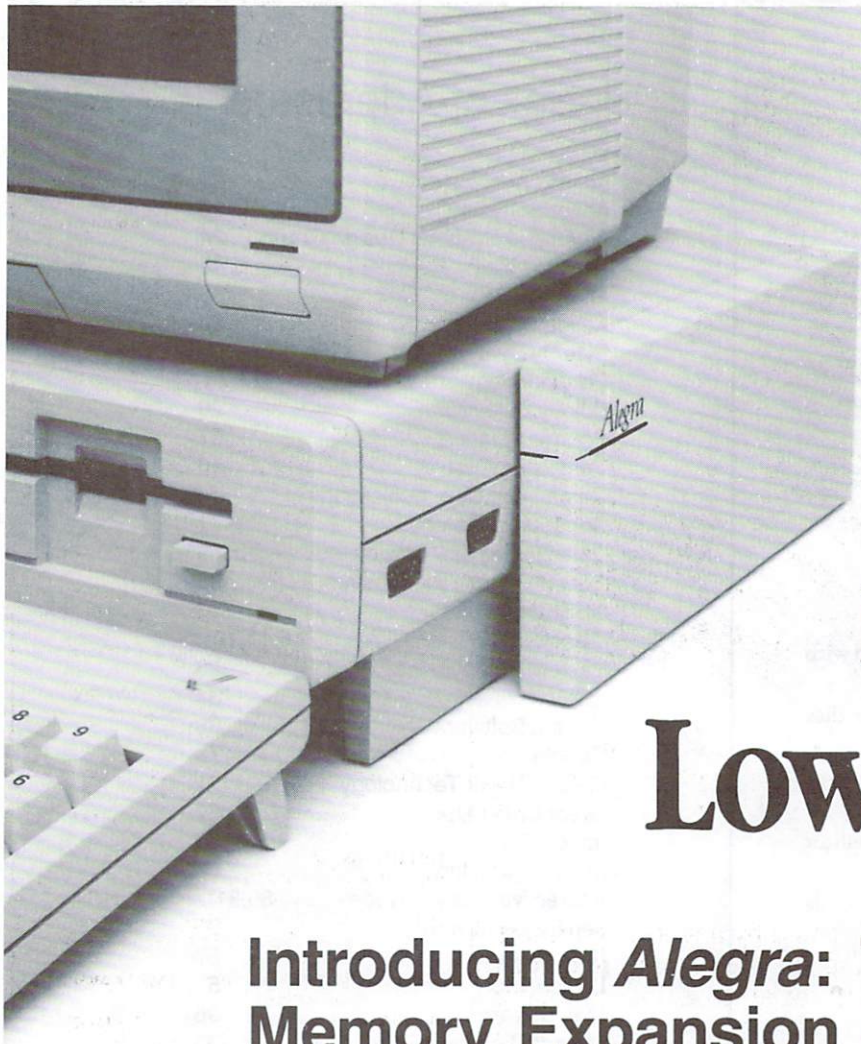
(Fred Fish Disk #30 is free, when ordered with at least three other disks from the collection.)

To Be Continued.....

In Conclusion

To the best of our knowledge, the materials in this library are freely redistributable. This means they were either publicly posted and placed in the Public Domain by their author, or they have restrictions published in their files to which we have adhered. If you become aware of any violation of the author's wishes, please contact us by mail.

•AC•



Smallest footprint. Lowest price.

Introducing *Alegra*: The Amiga™ Memory Expansion Unit from Access Associates.

***Alegra*: 512K now.**

Now you can add 512K of external memory to your Amiga. In the smallest package available. At the lowest price available: $339.00, suggested retail.

***Alegra E*: 512K now. 2 MB later.**

If you'll need 2 MB of memory in the future, the upgradeable Alegra E is the right choice now. Alegra E gives you 512K bytes of added memory with the option of 2 MB later.* Because the upgrade requires only internal component changes, Alegra maintains its slim ¾"-wide footprint. The price: $379.00, suggested retail.

Alegra features a 90 day parts and labor warranty against manufacturing defects.

See Alegra at your quality Amiga dealer.

Dealer inquiries invited.

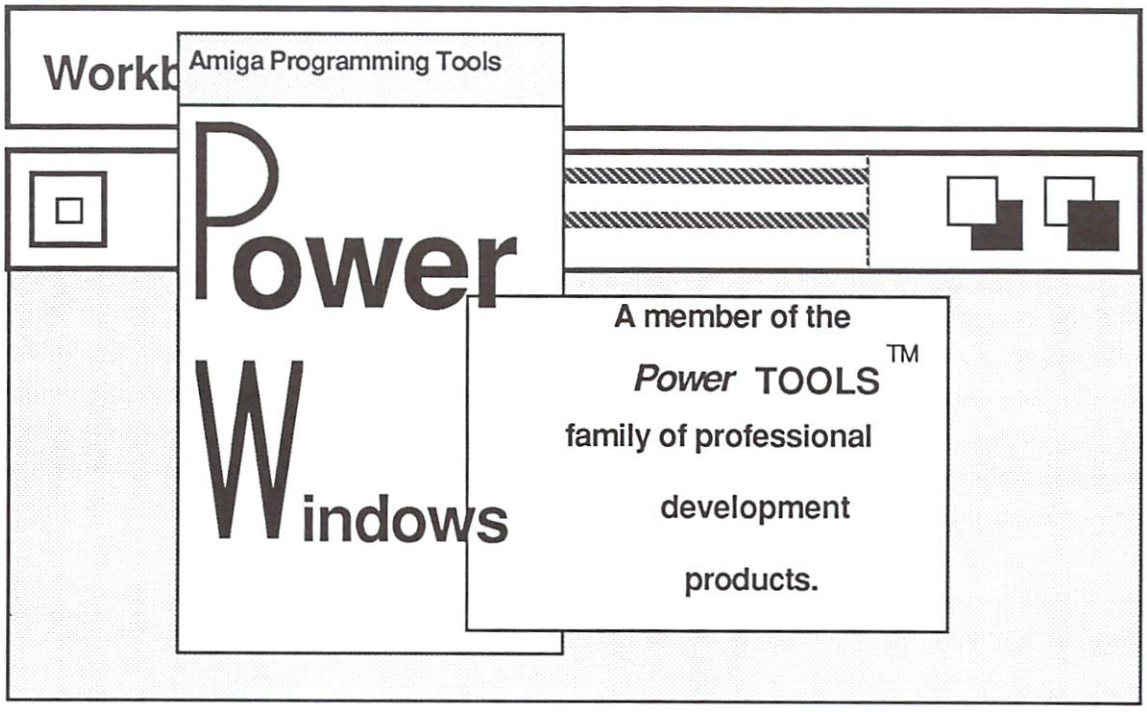
ACCESS ASSOCIATES

491 Aldo Avenue
Santa Clara, CA. 95054-2303
408-727-8520

**As 1 Mbit DRAMs become widely available.*

™Amiga is a trademark of Commodore Amiga, Inc.

Index of Adertisers



The **first** interactive Amiga program design tool, PowerWindows™ lets you to design fantastic looking windows, menus and gadgets in minutes instead of hours or days! You show this incredible program what you want and it does the rest, generating C or 68000 assembler source code for you to include in your own programs. PowerWindows is a structure generator for a machine that **thrives** on structures. With this software package you can:

Pick the exact size and position for your windows **visually.** No more "wait to see what it looks like"; PowerWindows knows where your window is and everything else about it!

Design professional looking menus. Add menus, move menus, or delete menus, whatever you want to do with text menus, our program keeps track of them and writes source code letting you duplicate them exactly with simple operating system calls.

Create your own string, integer and boolean gadgets and position them anywhere in your window. PowerWindows keeps them from colliding and remembers the type, location and text contents of each one for writing those complex gadget structures.

Best of all, you can keep your designs in a format that can be re-edited, letting you create your favorite type of windows and customize them for each program you write.

Order Form

Price for PowerWindows is $89.95, plus $3.50 for shipping and handling. Texas residents please add 6.125% sales tax to total price.

Name _____

Address _____

City _____ **State** _____ **Zip** _____

Products ordered _____

Payment method: _____ MC/Visa _____ Check _____ Money Order AC

Card Number _____ Expiration Date _____

Name on card _____ Signature _____

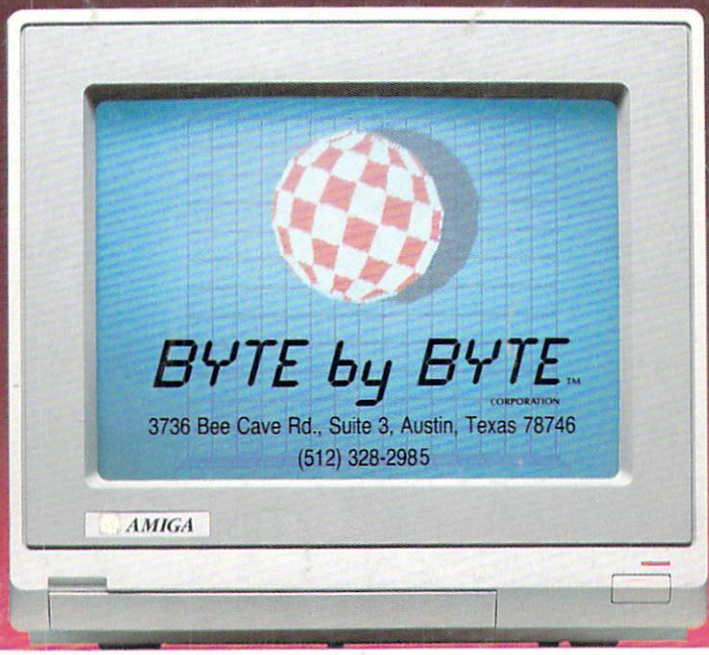
Enter total enclosed: _____

INOVATRONICS, INC.

11311 Stemmons Frwy., Suite 7 Dallas, TX 75229 214/241-9515

UNLEASH THE AWESOME POWER OF THE AMIGA!

The PAL is a turnkey expansion chassis that provides the most powerful and cost effective hardware growth path for your AMIGA. Features: High speed direct Amiga DMA controller and hard disk • Five DMA expansion slots • 1 Meg Ram with Clock/Calendar • Room for multiple storage/retrieval devices • 100% compatible with current and future Amigas • 1 to 8 megabyte ram card options • Optional pass through bus connector for further expansion • Optional prototyping card • Future products currently under development



INFOMINDER is an intelligent information resource that provides the user with instantaneous access to reference information stored within the Amiga personal computer.

Fully supports multi-tasking • Fast access by menu or outline • Text capabilities include: Justification, Word Wrap, Multiple character fonts/styles • Information content completely user definable • Supports combination of TEXT and IFF GRAPHICS • Programmatic interface for context sensitive help • Narration and printing of information • Expand and shrink topics.

INFOMINDER will revolutionize the way we access textual and graphical information. Stop searching and START using the information around you.

Special introductory price $89.95

WRITE HAND is a general word processor and form letter generator that gives you the most features for your dollars. Developed to meet the special needs of small business, WRITE HAND is easy to learn and easy to use.

WRITE HAND challenges you to compare the following features dollar-for-dollar, feature-for-feature to those of other word processors on the market today.

• Extensive on-line HELP service • Form letter generator • Powerful editing capabilities • Formats documents while you edit • Reviews and merges files while you edit • Moves blocks of text and figures of any size • Provides word wrap, bolding and underlining

Make WRITE HAND the tool that moves your business into the productive world of electronic word processing.

Suggested retail price $50.

FINANCIAL PLUS is the affordable way to put your business at your fingertips. FINANCIAL PLUS is the complete accounting solution with five systems in one:

• General Ledger • Accounts Payable • Accounts Receivable • Payroll • Word Processor

FINANCIAL PLUS is adaptable. You customize each company according to its size and bookkeeping needs.

An easy-to-read, easy-to-learn users guide provides comprehensive instructions for setting up your own books. Plain-English menus are the system "roadmaps" for both the novice and for the more experienced. Because FINANCIAL PLUS is a totally integrated accounting system, no longer must you purchase individual packages, store entries on separate diskettes, or run confusing transfer programs to obtain complete integration.

Suggested retail price $295.